Oracle® Database

Graph Developer's Guide for RDF Graph

23ai
F46994-14
March 2025

Oracle Database Graph Developer's Guide for RDF Graph, 23ai

F46994-14

Contributors: Lavanya Jayapalan

Contributors: Melliyal Annamalai , Maitreyee Chaliha, Chuck Murray, Eugene Inseok Chong, Souri Das, Joao Paiva, Matt Perry, Jags Srinivasan, Seema Sundara, Zhe (Alan) Wu, Aravind Yalamanchi

Contents

2 Quick Start for Using RDF Data

3 OWL Concepts

4 Simple Knowledge Organization System (SKOS) Support

5 Semantic Indexing for Documents

8 RDF Graph Support for Eclipse RDF4J

9 User-Defined Inferencing and Querying

14 Oracle RDF Graph Query UI

Part III Reference Information

15 SEM_APIS Package Subprograms

16 SEM_PERF Package Subprograms

Index

List of Figures

List of Tables

Preface

Oracle Database Graph Developer's Guide for RDF Graph provides usage and reference information about Oracle Database Enterprise Edition support for semantic technologies, including storage, inference, and query capabilities for data and ontologies based on Resource Description Framework (RDF), RDF Schema (RDFS), and Web Ontology Language (OWL). The RDF Graph feature is licensed with the Oracle Spatial and Graph option to Oracle Database Enterprise Edition, and it requires the Oracle Partitioning option to Oracle Database Enterprise Edition.

> **Note:**
>
> You must perform certain actions and meet prerequisites before you can use any types, synonyms, or PL/SQL packages related to RDF Graph support. These actions and prerequisites are explained in Enabling RDF Semantic Graph Support.

Audience

This guide is intended for those who need to use semantic technology to store, manage, and query semantic data in the database.

You should be familiar with at least the main concepts and techniques for the Resource Description Framework (RDF) and the Web Ontology Language (OWL).

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Related Documents

For an excellent explanation of RDF concepts, see the World Wide Web Consortium (W3C) *RDF Primer* at `http://www.w3.org/TR/rdf-primer/`.

For information about OWL, see the *OWL Web Ontology Language Reference* at `http://www.w3.org/TR/owl-ref/`.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Changes in This Release for This Guide

This topic contains the following.

- Changes in Oracle Database Release 23ai

Changes in Oracle Database Release 23ai

The following are the changes in *Oracle Database Graph Developer's Guide for RDF Graph* for Oracle Database Release 23ai.

Enhanced Support for Querying Semantic Data

In addition to the SEM_MATCH table function, you can also query semantic data in the following order:

- Use the SEM_APIS.GET_SQL function to obtain the SQL translation for a SPARQL query.
- Run the SEM_APIS.CREATE_SEM_SQL as a one-time setup procedure to create the SEM_SQL SQL Macro.
- Compile the SQL (SEM_APIS.SEM_SQL_COMPILE) and query the semantic data using the SEM_SQL SQL Macro.

See Using the SEM_APIS.GET_SQL Function and SEM_SQL SQL Macro to Query RDF Data for more information.

Support for In-Memory Subject-Property-Matrix Tables

You can create in-memory Subject-Property-Matrix (SPM) tables by using the `INMEMORY=T` option in SEM_APIS.BUILD_SPM_TAB.

See In-Memory Result Tables for more information.

Support for GeoSPARQL 1.1

The Open Geospatial Consortium (OGC) has proposed GeoSPARQL 1.1 as an update to the original OGC GeoSPARQL standard. This update includes new literal data types based on GeoJSON and KML geometry serializations and several new spatial query functions and spatial aggregates.

These new GeoSPARQL 1.1 geometry literals, query functions and aggregates can be used in SPARQL queries through SPARQL APIs provided by the RDF feature of Oracle Database.

See Spatial Support for more information.

Support for Auto-List Subpartitioning of RDF_LINK$ table

To improve the performance of SPARQL update (`CLEAR`, `MOVE`, `COPY`, or `DROP` query constructs, using keywords such as, `DEFAULT`, `NAMED`, `GRAPH`, and `ALL`), you can create the RDF_LINK$ table as a list-list composite partitioned table where subpartitions are automatically maintained on the graph ID. You can create the auto-list subpartitioned table by using the MODEL_PARTITIONING=BY_LIST_G option in SEM_APIS.CREATE_SEM_NETWORK.

See RDF Networks for more information.

Support for Retrieving Query Execution Plan Cost

You can use the SEM_APIS.GET_PLAN_COST API procedure to get the cost of a query execution plan.

See SEM_APIS.GET_PLAN_COST for more information.

Support for 32K VARCHAR RDF Values

You can now store RDF values up to 32767 bytes in length as VARCHAR values in your RDF network if your database has extended VARCHAR support enabled (MAX_STRING_SIZE=EXTENDED). In previous releases, only RDF values up to 4000 bytes in length were stored as VARCHAR values. RDF values larger than this limit (4K or 32K bytes), are stored as CLOBs. A 32K VARCHAR network results in less values being stored as CLOBs, which increases performance for queries, DMLs, and bulk load operations on large RDF literals.

To control the maximum VARCHAR size in your RDF network, you can pass NETWORK_MAX_STRING_SIZE=EXTENDED for 32K VARCHAR or NETWORK_MAX_STRING_SIZE=STANDARD (default) for 4K VARCHAR in the options argument of SEM_APIS.CREATE_RDF_NETWORK.

A pre-existing 4K VARCHAR RDF network cannot be migrated to a 32K VARCHAR RDF network. You must create a new RDF network using NETWORK_MAX_STRING_SIZE=EXTENDED and reload your data into the new network.

Deprecation of MDSYS-Owned RDF Network

Creation of RDF graph networks in the MDSYS schema is deprecated. Oracle recommends that you create RDF graph networks in a user schema, which was enabled in Oracle Database 19c.

An existing MDSYS-owned network can be migrated to a shared schema-private RDF network by using the SEM_APIS.MOVE_RDF_NETWORK_DATA and SEM_APIS.APPEND_RDF_NETWORK_DATA procedures.

See MDSYS-Owned Semantic Network in Appendix D for more information on MDSYS-owned semantic networks.

Running Graph Analytics Algorithms with RDF Graphs

You can create property graph views from an RDF graph. You can first run SEM_MATCH queries to create database views that represent vertex and edge tables, and then create a PGQL property graph from those views. This property graph can be loaded into the graph server for running graph analytics algorithms.

See Property Graph Views on RDF Graphs for more information.

How to Use This Book

This book is organized into three parts:

- Part I provides conceptual and usage information about RDF Graph.
- Part II provides information about using RDF Graph Server and Query UI.
- Part III provides reference information about RDF Graph subprograms.

All supplementary information is provided in Appendixes and specialized terms are defined in the Glossary.

However, the following summary provides an outline of some of the main ideas in the book that will help you to develop an understanding of RDF Graph support in Oracle Database and how to store, load, query, infer and visualize RDF data.

Learn About Oracle RDF Graph	**Get Started With Oracle RDF Graph**
Introduction to Oracle Semantic Technologies Support	Enabling RDF Graph Support
RDF Data in the Database	Quick Start for Using Semantic Data
OWL Concepts	Loading and Exporting RDF Data
RDF Views	Performing SPARQL Query operations
	Performing SPARQL Update operations
	Performance Tuning for SPARQL Queries
	Tuning the Performance of SPARQL Update Operations
What's New In Oracle RDF Graph	**Additional Oracle RDF Graph Features**
Speeding up Query Execution with Result Tables	RDF Graph Support for Apache Jena
RDF Graph Support for Eclipse RDF4J	RDF Support in SQL Developer
RDF Graph Server and Query UI	Using RDF with Oracle Database In-Memory
	Applying Oracle Machine Learning Algorithms to RDF Data

Part I

Conceptual and Usage Information

Part I provides conceptual and usage information about RDF Graph.

This part contains the following chapters:

- Property Graph Views on RDF Graphs
  Oracle Graph supports the property graph data model in addition to the RDF graph data model.

1 RDF Graph Overview

Oracle Graph support for semantic technologies consists mainly of Resource Description Framework (RDF) and a subset of the Web Ontology Language (OWL). These capabilities are referred to as the RDF Graph feature of Oracle Graph.

The RDF Graph feature enables you to create one or more RDF networks in an Oracle database. Each network contains RDF data.

This chapter assumes that you are familiar with the major concepts associated with RDF and OWL, such as {subject, predicate, object} triples, {subject, predicate, object, graph} quads, URIs, blank nodes, plain and typed literals, and ontologies. It does not explain these concepts in detail, but focuses instead on how the concepts are implemented in Oracle.

- For an excellent explanation of RDF concepts, see the World Wide Web Consortium (W3C) *RDF Primer* at `http://www.w3.org/TR/rdf-primer/`.
- For information about OWL, see the *OWL Web Ontology Language Reference* at `http://www.w3.org/TR/owl-ref/`.

The PL/SQL subprograms for working with RDF data are in the SEM_APIS package, which is documented in SEM_APIS Package Subprograms.

The RDF and OWL support are features of Oracle Graph, which must be installed for these features to be used. However, the use of RDF and OWL is not restricted to spatial data.

> **Note:**
>
> If you have any RDF data created using an Oracle Database release before 12.2, see Required Migration of Pre-12.2 RDF Data.

For information about OWL concepts and the Oracle Database support for OWL capabilities, see OWL Concepts .

> **Note:**
>
> Before performing any operations described in this guide, you must enable RDF Graph support in the database and meet other prerequisites, as explained in Enabling RDF Semantic Graph Support.

- Introduction to Oracle Semantic Technologies Support
  Oracle Database enables you to store RDF data and ontologies, to query RDF data and to perform ontology-assisted query of enterprise relational data, and to use supplied or user-defined inferencing to expand the power of querying on RDF data.
- Key Terms and Concepts for Working with RDF Graphs
  Learn the Oracle terminology and the concepts for working with the RDF graph feature in Oracle Database.

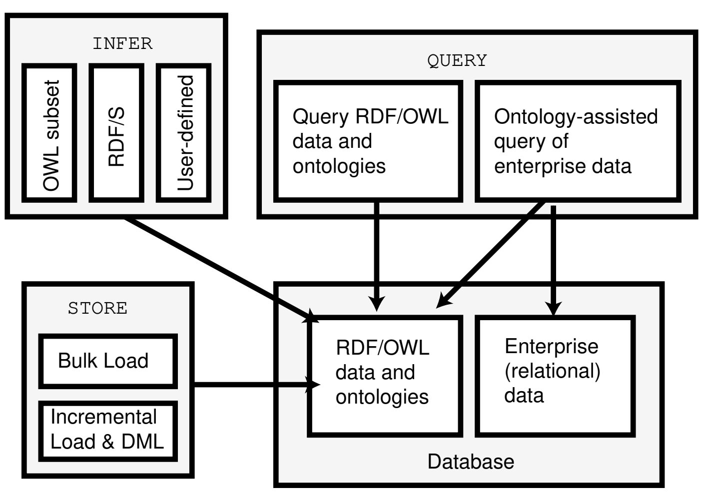
- Enhanced RDF ORDER BY Query Processing
  Effective with Oracle Database Release 12.2, queries on RDF data that use SPARQL ORDER BY semantics are processed more efficiently than in previous releases.
- Applying Oracle Machine Learning Algorithms to RDF Data
  You can apply Oracle Machine Learning algorithms to RDF data.
- RDF Graph Management Examples (PL/SQL and Java)
  PL/SQL examples are provided in this topic.
- Software Naming Changes Since Release 11.1
  Because the support for RDF data has been expanded beyond the original focus on RDF, the names of many software objects (PL/SQL packages, functions and procedures, system tables and views, and so on) have been changed as of Oracle Database Release 11.1.
- For More Information About RDF Graph
  More information is available about RDF graph support and related topics.
- Required Migration of Pre-12.2 RDF Data
  If you have any RDF data created using Oracle Database 11.1. 11.2, or 12.1, then before you use it in an Oracle Database 12.2 environment, you must migrate this data.
- Oracle RDF Graph Features that Support Accessibility
  This section describes the accessibility support provided by Oracle RDF Graph features.

1.1 Introduction to Oracle Semantic Technologies Support

Oracle Database enables you to store RDF data and ontologies, to query RDF data and to perform ontology-assisted query of enterprise relational data, and to use supplied or user-defined inferencing to expand the power of querying on RDF data.

Figure 1-1 shows how these capabilities interact.

Figure 1-1 Oracle RDF Capabilities



As shown in Figure 1-1, the database contains RDF data and ontologies (RDF/OWL graphs), as well as traditional relational data. To load RDF data, bulk loading is the most efficient approach, although you can load data incrementally using transactional INSERT statements.

Note:

If you want to use existing RDF data from a release before Oracle Database 11.1, the data must be upgraded as described in Enabling RDF Semantic Graph Support.

You can query RDF data and ontologies, and you can also perform ontology-assisted queries of RDF and traditional relational data to find semantic relationships. To perform ontology-assisted queries, use the SEM_RELATED operator, which is described in Using Semantic Operators to Query Relational Data.

You can expand the power of queries on RDF data by using inferencing, which uses rules in rulebases. Inferencing enables you to make logical deductions based on the data and the rules. For information about using rules and rulebases for inferencing, see Inferencing: Rules and Rulebases.

1.2 Key Terms and Concepts for Working with RDF Graphs

Learn the Oracle terminology and the concepts for working with the RDF graph feature in Oracle Database.

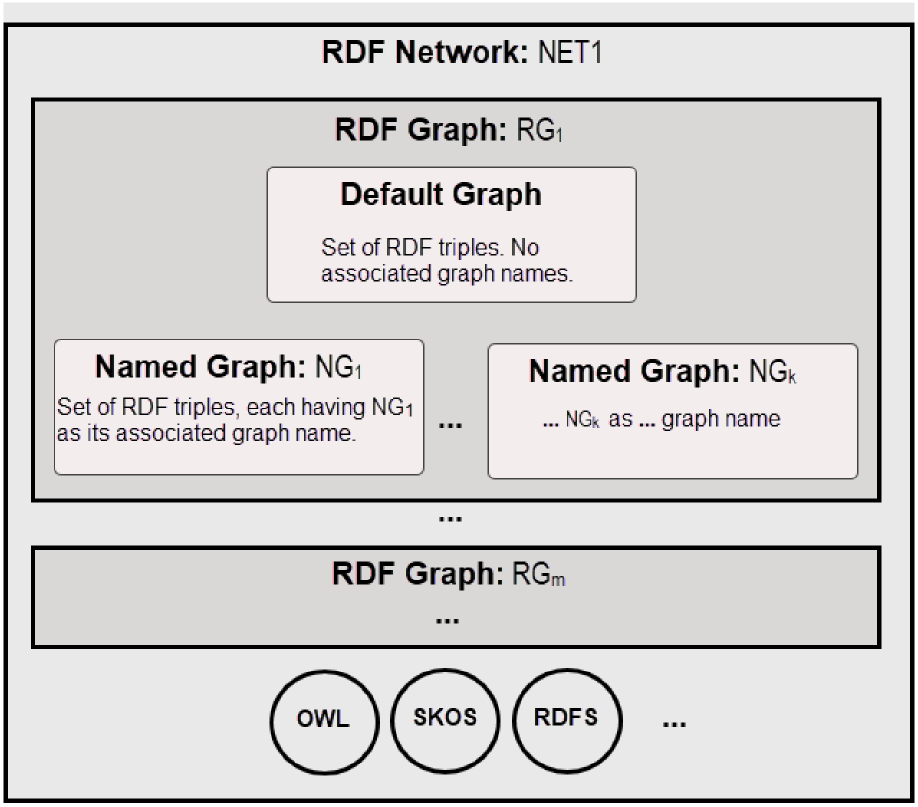
Although the terminology used in this guide for RDF concepts are very similar to the W3C RDF 1.1 terminology, there are some differences. The most significant difference is that the term **RDF Graph** in this document corresponds to **RDF Dataset** in W3C RDF 1.1 terminology.

The following table lists the Oracle RDF terminology and their corresponding mapping to the W3C RDF 1.1 terminology.

Table 1-1 Mapping Oracle and W3C RDF 1.1 Terms

Oracle Terminology	Description	W3C RDF 1.1 Terminology
RDF Network	Zero or more **RDF Graphs** and built-in (such as OWL, RDFS, and so on) and any user-defined **rulebases**.	None
RDF Graph	Single **Default Graph** and zero or more **Named Graphs**.	RDF Dataset
Default Graph	Set of triples that are not associated with any graph name. It is a part of an **RDF Graph**.	Default (RDF) Graph
Named Graph	Set of triples, each associated with the same graph name. Each named graph is part of an **RDF Graph**.	Named (RDF) Graph
RDF Graph Collection	Set of **RDF Graphs**.	Merged RDF Datasets
Inferred Graph	**RDF Graph** comprising only the triples inferred using specified **RDF Graphs** and **rulebases**.	Entailed Graph, but excludes the triples asserted in the dataset.
RDFview Graph	Relational data (in one or more tables) viewed as **RDF Graph** using W3C RDB2RDF mapping.	RDF Dataset obtained from relational tables using W3C RDB2RDF mapping.

The following diagram shows the structural representation of the Oracle RDF concepts described in the preceding table:



1.3 RDF Data Modeling

In addition to its formal semantics, RDF data has a simple data structure that is effectively modeled using a directed graph.

The metadata statements are represented as triples: nodes are used to represent two parts of the triple, and the third part is represented by a directed link that describes the relationship between the nodes. The triples are stored in an RDF data network. In addition, information is maintained about specific RDF graphs created by database users. A user-created **RDF Graph** has a graph name, and refers to triples stored in a specified table column.

Statements are expressed in triples: {subject or resource, predicate or property, object or value}. In this manual, {subject, property, object} is used to describe a triple, and the terms *statement* and *triple* may sometimes be used interchangeably. Each triple is a complete and unique fact about a specific domain, and can be represented by a link in a directed graph.

1.4 RDF Data in the Database

RDF data in Oracle Database is stored in one or more RDF networks.

All triples are parsed and stored in the system as entries in tables is an RDF network, and each RDF network is under a regular database user schema. A triple {subject, property, object} is treated as one database object. As a result, a single document containing multiple triples results in multiple database objects.

All the subjects and objects of triples are mapped to nodes in a RDF data network, and properties are mapped to network links that have their start node and end node as subject and object, respectively. The possible node types are blank nodes, URIs, plain literals, and typed literals.

The following requirements apply to the specifications of URIs and the storage of RDF data in the database:

- A subject must be a URI or a blank node.
- A property must be a URI.
- An object can be any type, such as a URI, a blank node, or a literal. (However, null values and null strings are not supported.)

1.4.1 RDF Networks

An **RDF network** is a set of tables and views that holds RDF data. An RDF network is not created during installation. A database user must explicitly call SEM_APIS.CREATE_RDF_NETWORK to create an RDF network before any RDF data can be stored in the database.

> **Note:**
>
> **RDF Networks** were called as **Semantic Networks** in the previous book versions (prior to Oracle Database Release 23ai). See Changes in Terminology and Subprograms for more information.

An RDF network contains, among other things, an RDF_LINK$ table for storing RDF triples or quads. By default, the RDF_LINK$ table is list-partitioned into a set of RDF Graphs, which are user-created containers for storing RDF triples or quads.

The RDF_LINK$ table can optionally use list-hash composite partitioning, where each RDF graph partition is subpartitioned by a hash of the predicate. Composite partitioning can improve SPARQL query performance on larger data sets through better parallelization and improved query optimizer statistics.

The RDF_LINK$ table can also optionally use list-list composite partitioning, where each RDF graph partition is subpartitioned by the graph `ID`. The subpartition is automatically maintained on the graph `ID`. This configuration is highly recommended for quad data as it will drastically improve the performance of SPARQL update (`CLEAR`, `MOVE`, `DROP`, or `COPY`) graph operations.

For more information about how to enable composite partitioning, see:

- The `options` parameter descriptions for SEM_APIS.CREATE_RDF_GRAPH and SEM_APIS.CREATE_RDF_NETWORK.
- The usage notes for the `options` parameter for SEM_APIS.CREATE_INFERRED_GRAPH, specifically for the `MODEL_PARTITIONS=n` option.

An RDF_VALUE$ table is used to store a mapping of RDF values to internal numeric identifiers. Starting with version 21c, values stored in the RDF_VALUE$ table can be stored using an unescaped storage form; that is, Unicode characters and special characters are stored as a single character instead of being stored as an ASCII escape sequence (such as the single character 'ñ' instead of the ASCII escape sequence '\u00F1'). This unescaped storage form reduces storage costs and increases query performance.

The network storage form can be specified in the `options` parameter of the SEM_APIS.CREATE_RDF_NETWORK procedure at network creation time. Unescaped storage form is the default in version 21c and later. Existing RDF networks can be migrated using the SEM_APIS.MIGRATE_DATA_TO_STORAGE_V2 procedure.. Existing applications should not be affected by any changes in network storage form.

Starting with Oracle Database 23ai, the following are the two options for the maximum size of VARCHAR values stored in RDF_VALUE$ table:

- **4000 bytes:** This is the default maximum size.
- **32767 bytes:** If the database has extended VARCHAR enabled (see Extended Data Types), then the default maximum size can optionally be extended to 32767 bytes.

RDF values smaller than or equal to this maximum size of 4K or 32K will be stored as VARCHARs, and larger values will be stored as CLOBs. Using a 32K maximum VARCHAR size results in fewer values being stored as CLOBs, which increases performance of query, DML, and bulk load of large RDF values.

The maximum VARCHAR size for a network can be specified in the `options` parameter of the SEM_APIS.CREATE_RDF_NETWORK procedure at network creation time.

- **`NETWORK_MAX_STRING_SIZE=STANDARD`:** Indicates a maximum size of 4000 bytes and is the default.
- **`NETWORK_MAX_STRING_SIZE=EXTENDED`:** Indicates a maximum size of 32767 bytes.

The `NETWORK_MAX_STRING_SIZE` setting for an RDF network is recorded in the network's `RDF_PARAMETER` table.

One or more RDF networks can be created and owned by a regular database user schema. Each such network is called a **schema-private RDF network**. You can have such a network in a single database or pluggable database.

> **Note:**
>
> Starting from Oracle Database Release 23ai, MDSYS-owned RDF networks are deprecated. It is recommended that you create schema-private RDF networks. An existing MDSYS-owned RDF network can be migrated to a shared schema-private RDF network by using the SEM_APIS.MOVE_RDF_NETWORK_DATA and SEM_APIS.APPEND_RDF_NETWORK_DATA procedures. Also, see Moving, Restoring, and Appending an RDF Network for more information.

- Schema-Private RDF Networks

- Types of RDF Network Users
- Naming Conventions for RDF Network Objects
- RDF_PARAMETER Table in RDF Networks
- Migrating from MDSYS to Schema-Private RDF Networks
- Sharing Schema-Private RDF Networks
- Migrating from Escaped to Unescaped Storage Form

1.4.1.1 Schema-Private RDF Networks

In a schema-private RDF network, the associated database objects are created in the network owner's schema, and the network owner has exclusive privileges to those objects. (DBA users also have such privileges, and the network owner or a DBA can grant and revoke the privileges for other users.)

Schema-private RDF networks have several benefits:

- They provide better security and isolation because multiple users do not share tables and indexes.

  The network owner's schema contains all RDF network database objects, and the network owner has exclusive privileges to those objects by default.

  Schema-private RDF networks provide better isolation because database objects are not shared among multiple database users by default. However, after granting appropriate privileges, a network owner may share their schema-private RDF network with other users.

- Regular users can perform administration operations on their own networks, for example, index creation or network-wide statistics gathering.

  The network owner can perform administration operations on the network without needing DBA privileges.

  Several schema-private RDF networks can coexist in a single database, PDB, or even schema, which allows custom data type indexing schemes for different sets of RDF data. For example, NETWORK1 can have only a spatial data type index while NETWORK2 has only a text data type index.

Most SEM_APIS package subprograms now have `network_owner` and `network_name` parameters to support schema-private RDF networks. Schema-private RDF networks are identified by the two-element combination of network owner and network name, which is specified in the last two parameters of the SEM_APIS.CREATE_SEM_NETWORK call that created the network.

The following table describes the usage of the `network_owner` and `network_name` parameters in subprograms that include them.

Table 1-2 network_owner and network_name Parameters

Parameter Name	Description
`network_owner`	Name of the schema that owns the network. The network owner must be a non-NULL value that specifies a regular database user.

Table 1-2 (Cont.) network_owner and network_name Parameters

Parameter Name	Description
`network_name`	Name of the network. • The network name must be a non-NULL value. • The network name must be unique within the schema of the network owner. For example, schema SCOTT cannot have two networks named NET1; but schemas SCOTT and ANNA can each have a network named NET1.

1.4.1.2 Types of RDF Network Users

The following three key types of RDF network users are supported:

- **Network Creator:** A user that invokes SEM_APIS.CREATE_RDF_NETWORK. The network creator is either a database user with DBA privileges or it is the same as the network owner.
- **Network Owner:** A user whose schema will hold the tables, triggers and views that make up the RDF network.
- **Network User:** A database user that performs operations on the RDF network.
  In many examples in this book, the name `RDFUSER` is given as a sample network user name. There is nothing special about that name string; it could be the name of any database user such as `SCOTT`, `ANNA`, or `MARKETING`.

  The network owner is initially the only network user. (However, other database users can be granted privileges on the network, thus making them additional potential network users.)

1.4.1.3 Naming Conventions for RDF Network Objects

RDF network database objects follow specific naming conventions.

All RDF network database objects in a schema-private network are prefixed with *NETWORK_NAME#*, for example, USER3.MYNET#SEM_MODEL$. This book uses the portion of the database object name after the prefix to refer to the object. That is, SEM_MODEL$ refers to *NETWORK_OWNER.NETWORK_NAME#*SEM_MODEL$ for a schema-private RDF network.

1.4.1.4 RDF_PARAMETER Table in RDF Networks

The MDSYS.RDF_PARAMETER table holds database-wide RDF Graph installation information. This table is created during installation and always exists.

In schema-private RDF networks, a *NETWORK_NAME#*RDF_PARAMETER table holds network-specific information such as network compression settings and any RDFCTX or RDFOLS policies used in the schema-private network.

A schema-private *NETWORK_NAME#*RDF_PARAMETER table is dependent on the existence of the *NETWORK_NAME* RDF network. This table is created during schema-private RDF network creation and is dropped when the schema-private network is dropped.

1.4.1.5 Migrating from MDSYS to Schema-Private RDF Networks

An existing MDSYS-owned RDF network can be migrated to a shared schema-private RDF network by using the SEM_APIS.MOVE_RDF_NETWORK_DATA and

SEM_APIS.APPEND_RDF_NETWORK_DATA procedures. See Migrating an MDSYS-Owned Network to a Schema-Private Network for details.

1.4.1.6 Sharing Schema-Private RDF Networks

After a schema-private network is created, it can optionally be shared, that is, made available for use by other database users besides the network owner. Other users can be allowed to have either of the following access capabilities:

- Read-only access to RDF data, which provides the ability to query the RDF data in the network.
  Granting read-only or query-only access to an RDF network can be done by:
  1. The network owner by using the single command SEM_APIS.GRANT_NETWORK_ACCESS_PRIVS with `QUERY_ONLY=T` included in the `OPTIONS` parameter.
  2. The network owner or the RDF graph owner by using SEM_APIS.GRANT_RDF_GRAPH_ACCESS_PRIVS with appropriate privileges such as `QUERY` or `SELECT` for the individual RDF graphs in the network.

  See Example 1-1 for more details.
- Read/write access to RDF objects and data in the network, such as the ability to create, alter, or drop RDF graphs and inferred graphs, and to read, insert, modify, or delete RDF data.
  The logical sequence of steps for granting both read and write access is as follows:
  1. A DBA must grant network sharing privileges to the network owner. This needs to be done only once for a given network owner. However, you can skip this step if you are using Oracle Database version 23.4.
  2. The network owner must enable the specific network for sharing. This needs to be done only once for a given network.
  3. The network owner must grant network access privileges to the user(s) that will be allowed to access the network.
     Each of these grants can subsequently be revoked, if necessary.

  See Example 1-2 for more details.

> **Note:**
>
> Having the above access capabilities for a network allows a user to access only the dictionary and metadata tables for the network. RDF graphs and inferred graphs not owned by the user are not accessible unless the network owner or the owner of the individual RDf graphs use the SEM_APIS.REVOKE_RDF_GRAPH_ACCESS_PRIV or SEM_APIS.GRANT_RDF_GRAPH_ACCESS_PRIVS subprogram to grant appropriate privilege(s) for individual RDF graphs or inferred graphs in the network to the user.

Example 1-1 Sharing a Network and Granting Query Only Privilege to Another User

The following example shares a network named NET1, owned by user RDFUSER. RDFUSER grants query-only access on NET1 with user RDFQ.

```
-- As RDFUSER, create a schema-private network owned by RDFUSER named NET1
CONNECT rdfuser/<password>;
```

```
EXECUTE
SEM_APIS.CREATE_RDF_NETWORK('RDFTBS',network_owner=>'RDFUSER',network_name=>'N
ET1');

-- As RDFUSER, grant query only network access privilege for NET1 to RDFQ
EXECUTE
SEM_APIS.GRANT_NETWORK_ACCESS_PRIVS(network_owner=>'RDFUSER',network_name=>'NE
T1',network_user=>'RDFQ', options=>' QUERY_ONLY=T ');

-- As RDFUSER, create an RDF graph M1 in network NET1
EXECUTE
SEM_APIS.CREATE_RDF_GRAPH('M1',null,null,network_owner=>'RDFUSER',network_name
=>'NET1');

-- Check metadata
SELECT *
FROM rdfuser.net1#sem_model$;

-- Insert some data
INSERT INTO rdfuser.net1#rdft_m1(triple)
VALUES
(SDO_RDF_TRIPLE_S('M1','<urn:person1>','<urn:name>','"Peter"','RDFUSER','NET1'
));
COMMIT;

-- Allow RDFQ to select and query an RDF graph that RDFUSER owns
EXECUTE
SEM_APIS.GRANT_RDF_GRAPH_ACCESS_PRIVS('M1','RDFQ',sys.odcivarchar2list('SELECT
','QUERY'),network_owner=>'RDFUSER',network_name=>'NET1');

-- As RDFQ, verify that RDF graph M1 is visible for querying
CONNECT rdfq/<password>;
SELECT *
FROM rdfuser.net1#rdf_model$
WHERE model_name='M1';

-- Query with SEM_MATCH
SELECT s$rdfterm, p$rdfterm, o$rdfterm
FROM TABLE(SEM_MATCH(
'SELECT ?s ?p ?o
 WHERE { ?s ?p ?o }'
,SEM_MODELS('M1')
,null,null,null,null
,' PLUS_RDFT=VC '
,null,null
,'RDFUSER','NET1'));
```

Example 1-2 Sharing a Network and Granting Read and Write Privileges to Another User

The following example shares a network named NET1, owned by user RDFUSER, with user RDFUSER2. Also RDFUSER grants query-only access on NET1 with user RDFUSER3.

```
-- As RDFUSER, create a schema-private network owned by RDFUSER named NET1
CONNECT rdfuser/<password>;
EXECUTE
```

```
SEM_APIS.CREATE_RDF_NETWORK('RDFTBS',network_owner=>'RDFUSER',network_name=>'N
ET1');

-- As RDFUSER, enable sharing for NET1
CONNECT rdfuser/<password>;
EXECUTE
SEM_APIS.ENABLE_NETWORK_SHARING(network_owner=>'RDFUSER',network_name=>'NET1')
;

-- As RDFUSER, grant network access privileges for NET1 to RDFUSER2
EXECUTE
SEM_APIS.GRANT_NETWORK_ACCESS_PRIVS(network_owner=>'RDFUSER',network_name=>'NE
T1',network_user=>'RDFUSER2');

-- As RDFUSER2, create a RDF graph M2 in network NET1
CONNECT rdfuser2/<password>;
EXECUTE
SEM_APIS.CREATE_RDF_GRAPH('M2',null,null,network_owner=>'RDFUSER',network_name
=>'NET1');

-- Check metadata
SELECT *
FROM rdfuser.net1#sem_model$;

-- Insert some data
INSERT INTO rdfuser.net1#rdft_m2(triple)
VALUES
(SDO_RDF_TRIPLE_S('M2','<urn:person1>','<urn:name>','"John"','RDFUSER','NET1')
);
COMMIT;

-- Query with SEM_MATCH
SELECT s$rdfterm, p$rdfterm, o$rdfterm
FROM TABLE(SEM_MATCH(
'SELECT ?s ?p ?o
 WHERE { ?s ?p ?o }'
,SEM_MODELS('M2')
,null,null,null,null
,' PLUS_RDFT=VC '
,null,null
,'RDFUSER','NET1'));

-- As RDFUSER, grant query only network access privileges for NET1 to RDFUSER3
CONNECT rdfuser/<password>
EXECUTE
SEM_APIS.GRANT_NETWORK_ACCESS_PRIVS(network_owner=>'RDFUSER',network_name=>'NE
T1',network_user=>'RDFUSER3', options=>' QUERY_ONLY=T ');

-- As RDFUSER2, allow RDFUSER3 to select and query an RDF graph that RDFUSER2
owns
CONNECT rdfuser2/<password>
EXECUTE
SEM_APIS.GRANT_RDF_GRAPH_ACCESS_PRIVS('M2','RDFUSER3',sys.odcivarchar2list('SE
LECT','QUERY'),network_owner=>'RDFUSER',network_name=>'NET1');

-- As RDFUSER3, verify that RDF graph M2 is visible for querying
```

```
CONNECT rdfuser3/<password>
SELECT *
FROM rdfuser.net1#rdf_model$
WHERE model_name='M2';

-- Query with SEM_MATCH
SELECT s$rdfterm, p$rdfterm, o$rdfterm
FROM TABLE(SEM_MATCH(
'SELECT ?s ?p ?o
 WHERE { ?s ?p ?o }'
,SEM_MODELS('M2')
,null,null,null,null
,' PLUS_RDFT=VC '
,null,null
,'RDFUSER','NET1'));
```

1.4.1.7 Migrating from Escaped to Unescaped Storage Form

You can migrate an existing RDF network from escaped storage form to unescaped storage form by using the SEM_APIS.MIGRATE_DATA_TO_STORAGE_V2 procedure. This procedure must be called by a DBA or the network owner.

Note that migration in the reverse direction is not possible. That is, you cannot migrate an RDF network from unescaped storage form to escaped storage form.

1.4.2 RDF Graphs

An **RDF graph** is a user-created container for storing RDF triples or quads. An RDF network contains zero or more RDF graphs. You can use the SEM_APIS.CREATE_RDF_GRAPH procedure to create an RDF graph. Each graph is physically stored as a partition in the network's RDF_LINK$ table. Besides the corresponding RDF_LINK$ partition, each graph is associated with two other database objects.

> **Note:**
>
> **RDF graphs** were called as **Semantic Models** in the previous book versions (prior to Oracle Database Release 23ai). See Changes in Terminology and Subprograms for more information.

In a ***schema-private*** RDF network, each graph is associated with (1) a SEMM_<rdf_graph_name> view of the graph's RDF_LINK$ partition, and (2) an RDFT_<rdf_graph_name> application view for the graph.

The application view is created automatically in the network owner's schema and has one column named TRIPLE with type SDO_RDF_TRIPLE_S. It is an updatable view that can be used to perform SQL DMLs on the associated RDF graph. The graph owner is given SELECT, INSERT, UPDATE, and DELETE privileges WITH GRANT OPTION on RDFT_<rdf_graph_name>.

You can truncate a graph using SEM_APIS.TRUNCATE_RDF_GRAPH.

The SEM_MODEL$ view contains information about all RDF graphs defined in an RDF network. When you create a graph using the SEM_APIS.CREATE_RDF_GRAPH procedure, you specify a name for the graph, as well as a table and column to hold references to the RDF data, and the system automatically generates a graph ID.

Oracle maintains the SEM_MODEL$ view automatically when you create and drop graphs. Users should never modify this view directly. For example, do not use SQL INSERT, UPDATE, or DELETE statements with this view.

The SEM_MODEL$ view contains the columns shown in Table 1-3.

Table 1-3 SEM_MODEL$ View Columns

Column Name	Data Type	Description
OWNER	VARCHAR2(30)	Schema of the owner of the RDF graph.
MODEL_ID	NUMBER	Unique model ID number, automatically generated.
MODEL_NAME	VARCHAR2(25)	Name of the RDF graph.
TABLE_NAME	VARCHAR2(30)	This value will be NULL for a schema-private network.
COLUMN_NAME	VARCHAR2(30)	This value will be NULL for a schema-private network.
MODEL_TABLESPACE_NAME	VARCHAR2(30)	Name of the tablespace to be used for storing the triples for this RDF graph.
MODEL_TYPE	VARCHAR2(40)	A value indicating the type of graph: `M` for regular RDF graph; `V` for RDF graph collection; `X` for RDF graph created to store the contents of the semantic index; or `D` for RDF graph created on relational data.
INMEMORY	VARCHAR2(1)	String value indicating if the i is an Oracle Database In-Memory RDF graph collection: `T` for in-memory, or `F` for not in-memory.

When you create an RDF graph, a view for the triples associated with the graph is also created under the network owner's schema. This view has a name in the format SEMM_<rdf_graph_name>, and it is visible only to the owner of the graph and to users with suitable privileges. Each SEMM_<rdf_graph_name> view contains a row for each triple (stored as a link in a network), and it has the columns shown in Table 1-4.

Table 1-4 SEMM_<rdf_graph_name> View Columns

Column Name	Data Type	Description
P_VALUE_ID	NUMBER	The VALUE_ID for the text value of the predicate of the triple. Part of the primary key.
START_NODE_ID	NUMBER	The VALUE_ID for the text value of the subject of the triple. Also part of the primary key.
CANON_END_NODE_ID	NUMBER	The VALUE_ID for the text value of the canonical form of the object of the triple. Also part of the primary key.
END_NODE_ID	NUMBER	The VALUE_ID for the text value of the object of the triple
MODEL_ID	NUMBER	The ID for the RDF graph to which the triple belongs.
COST	NUMBER	(Reserved for future use)
CTXT1	NUMBER	(Reserved column; can be used for fine-grained access control)
CTXT2	VARCHAR2(4000)	(Reserved for future use)
DISTANCE	NUMBER	(Reserved for future use)

Table 1-4 (Cont.) SEMM_<rdf_graph_name> View Columns

Column Name	Data Type	Description
EXPLAIN	VARCHAR2(4000)	(Reserved for future use)
PATH	VARCHAR2(4000)	(Reserved for future use)
G_ID	NUMBER	The VALUE_ID for the text value of the graph name for the triple. Null indicates the default graph (see Named Graphs).
LINK_ID	VARCHAR2(71)	Unique triple identifier value. (It is currently a computed column, and its definition may change in a future release.)

> **Note:**
>
> In Table 1-4, for columns P_VALUE_ID, START_NODE_ID, END_NODE_ID, CANON_END_NODE_ID, and G_ID, the actual ID values are computed from the corresponding lexical values. However, a lexical value may not always map to the same ID value.

1.4.3 Statements

The RDF_VALUE$ table contains information about the subjects, properties, and objects used to represent RDF statements. It uniquely stores the text values (URIs or literals) for these three pieces of information, using a separate row for each part of each triple.

Oracle maintains the RDF_VALUE$ table automatically. Users should never modify this view directly. For example, do not use SQL INSERT, UPDATE, or DELETE statements with this view.

The RDF_VALUE$ table contains the columns shown in Table 1-5.

Table 1-5 RDF_VALUE$ Table Columns

Column Name	Data Type	Description
VALUE_ID	NUMBER	Unique value ID number, automatically generated.
VALUE_TYPE	VARCHAR2(10)	The type of text information stored in the VALUE_NAME column. Possible values: `UR` for URI, `BN` for blank node, `PL` for plain literal, `PL@` for plain literal with a language tag, `PLL` for plain long literal, `PLL@` for plain long literal with a language tag, `TL` for typed literal, or `TLL` for typed long literal. A long literal is a literal with more than 4000 bytes.
VNAME_PREFIX	VARCHAR2(`NETWORK_MAX_STRING_SIZE`)	If the length of the lexical value is `NETWORK_MAX_STRING_SIZE` bytes or less, this column stores a prefix of a portion of the lexical value. The SEM_APIS.VALUE_NAME_PREFIX function can be used for prefix computation. For example, the prefix for the portion of the lexical value `<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>` without the angle brackets is `http://www.w3.org/1999/02/22-rdf-syntax-ns#`.

Table 1-5 (Cont.) RDF_VALUE$ Table Columns

Column Name	Data Type	Description
VNAME_SUFFIX	VARCHAR2(512)	If the length of the lexical value is `NETWORK_MAX_STRING_SIZE` bytes or less, this column stores a suffix of a portion of the lexical value. The SEM_APIS.VALUE_NAME_SUFFIX function can be used for suffix computation. For the lexical value mentioned in the description of the VNAME_PREFIX column, the suffix is `type`.
LITERAL_TYPE	VARCHAR2(4000)	For typed literals, the type information; otherwise, null. For example, for a row representing a creation date of 1999-08-16, the VALUE_TYPE column can contain `TL`, and the LITERAL_TYPE column can contain `http://www.w3.org/2001/XMLSchema#date`.
LANGUAGE_TYPE	VARCHAR2(80)	Language tag (for example, `fr` for French) for a literal with a language tag (that is, if VALUE_TYPE is `PL@` or `PLL@`). Otherwise, this column has a null value.
CANON_ID	NUMBER	The ID for the canonical lexical value for the current lexical value. (The use of this column may change in a future release.)
COLLISION_EXT	VARCHAR2(64)	Used for collision handling for the lexical value. (The use of this column may change in a future release.)
CANON_COLLISION_EXT	VARCHAR2(64)	Used for collision handling for the canonical lexical value. (The use of this column may change in a future release.)
ORDER_TYPE	NUMBER	Represents order based on data type. Used to improve performance on ORDER BY queries.
ORDER_NUM	NUMBER	Represents order for number type. Used to improve performance on ORDER BY queries.
ORDER_DATE	TIMESTAMP WITH TIME ZONE	Represents order based on date type Used to improve performance on ORDER BY queries.
LONG_VALUE	CLOB	The character string if the length of the lexical value is greater than `NETWORK_MAX_STRING_SIZE` bytes. Otherwise, this column has a null value.
GEOM	SDO_GEOMETRY	A geometry value when a spatial index is defined.
VALUE_NAME	VARCHAR2(`NETWORK_MAX_STRING_SIZE`)	This is a computed column. If length of the lexical value is `NETWORK_MAX_STRING_SIZE` bytes or less, the value of this column is the concatenation of the values of VNAME_PREFIX column and the VNAME_SUFFIX column.

- Triple Uniqueness and Data Types for Literals

1.4.3.1 Triple Uniqueness and Data Types for Literals

Duplicate triples are not stored in an RDF network. To check if a triple is a duplicate of an existing triple, the subject, property, and object of the incoming triple are checked against triple values in the specified RDF graph. If the incoming subject, property, and object are all URIs, an exact match of their values determines a duplicate. However, if the object of incoming triple is a literal, an exact match of the subject and property, and a value (canonical) match of the object, determine a duplicate. For example, the following two triples are duplicates:

```
<eg:a> <eg:b> <"123"^^http://www.w3.org/2001/XMLSchema#int>
<eg:a> <eg:b> <"123"^^http://www.w3.org/2001/XMLSchema#unsignedByte>
```

The second triple is treated as a duplicate of the first, because `"123"^^<http://www.w3.org/2001/XMLSchema#int>` has an equivalent value (is canonically equivalent) to `"123"^^<http://www.w3.org/2001/XMLSchema#unsignedByte>`. Two entities are canonically equivalent if they can be reduced to the same value.

To use a non-RDF example, `A*(B-C)`, `A*B-C*A`, `(B-C)*A`, and `-A*C+A*B` all convert into the same canonical form.

> **Note:**
>
> Although duplicate triples and quads are not stored in the underlying table partition for the RDFM_<*rdf_graph_name*> view, it is possible to have duplicate rows in an application table. For example, if a triple is inserted multiple times into an application table, it will appear once in the RDFM_<*rdf_graph_name*> view, but will occupy multiple rows in the application table.

Value-based matching of lexical forms is supported for the following data types:

- STRING: plain literal, xsd:string and some of its XML Schema subtypes
- NUMERIC: xsd:decimal and its XML Schema subtypes, xsd:float, and xsd:double. (Support is not provided for float/double INF, -INF, and NaN values.)
- DATETIME: xsd:datetime, with support for time zone. (Without time zone there are still multiple representations for a single value, for example, `"2004-02-18T15:12:54"` and `"2004-02-18T15:12:54.0000"`.)
- DATE: xsd:date, with or without time zone
- OTHER: Everything else. (No attempt is made to match different representations).

Canonicalization is performed when the time zone is present for literals of type xsd:time and xsd:dateTime.

The following namespace definition is used: `xmlns:xsd="http://www.w3.org/2001/XMLSchema"`

The first occurrence of a long literal in the RDF_VALUE$ table is taken as the canonical form and given the VALUE_TYPE value of `CPLL`, `CPLL@`, or `CTLL` as appropriate; that is, a `C` for canonical is prefixed to the actual value type. If a long literal with the same canonical form (but a different lexical representation) as a previously inserted long literal is inserted into the RDF_VALUE$ table, the VALUE_TYPE value assigned to the new insertion is `PLL`, `PLL@`, or `TLL` as appropriate.

Canonically equivalent text values having different lexical representations are thus stored in the RDF_VALUE$ table; however, canonically equivalent triples are not stored in the database.

1.4.4 Subjects and Objects

RDF subjects and objects are mapped to nodes in an RDF network. Subject nodes are the start nodes of links, and object nodes are the end nodes of links. Non-literal nodes (that is, URIs and blank nodes) can be used as both subject and object nodes. Literals can be used only as object nodes.

1.4.5 Blank Nodes

Blank nodes can be used as subject and object nodes in the RDF network. Blank node identifiers are different from URIs in that they are scoped within an RDF graph. Thus, although multiple occurrences of the same blank node identifier within a single RDF graph necessarily refer to the same resource, occurrences of the same blank node identifier in two different RDF graphs do not refer to the same resource.

In an Oracle RDF network, this behavior is modeled by requiring that blank nodes are always reused (that is, are used to represent the same resource if the same blank node identifier is used) within an RDF graph, and never reused between two different RDF graphs. Thus, when inserting triples involving blank nodes into an RDF graph, you must use the SDO_RDF_TRIPLE_S constructor that supports reuse of blank nodes.

1.4.6 Properties

Properties are mapped to links that have their start node and end node as subjects and objects, respectively. Therefore, a link represents a complete triple.

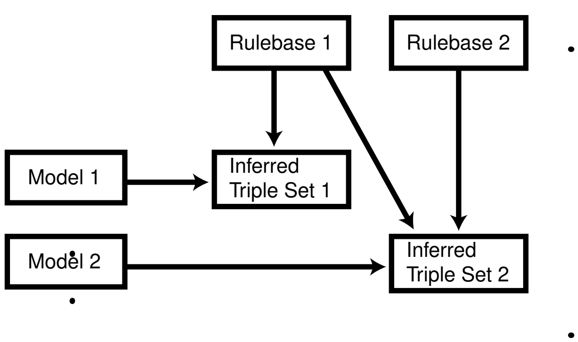
When a triple is inserted into an RDF graph, the subject, property, and object text values are checked to see if they already exist in the database. If they already exist (due to previous statements in other RDF graphs), no new entries are made; if they do not exist, three new rows are inserted into the RDF_VALUE$ table (described in Statements).

1.4.7 Inferencing: Rules and Rulebases

Inferencing is the ability to make logical deductions based on rules. Inferencing enables you to construct queries that perform semantic matching based on meaningful relationships among pieces of data, as opposed to just syntactic matching based on string or other values. Inferencing involves the use of rules, either supplied by Oracle or user-defined, placed in rulebases.

Figure 1-2 shows triple sets being inferred from RDF graph data and the application of rules in one or more rulebases. In this illustration, the database can have any number of RDF graphs, rulebases, and inferred triple sets, and an inferred triple set can be derived using rules in one or more rulebases.

Figure 1-2 Inferencing



A **rule** is an object that can be applied to draw inferences from RDF data. A rule is identified by a name and consists of:

- An IF side pattern for the antecedents
- A THEN side pattern for the consequents

For example, the rule that *a chairperson of a conference is also a reviewer of the conference* could be represented as follows:

```
('chairpersonRule', -- rule name
 '(?r :ChairPersonOf ?c)', -- IF side pattern
 NULL,  -- filter condition
 '(?r :ReviewerOf ?c)', -- THEN side pattern
 SEM_ALIASES (SEM_ALIAS('', 'http://some.org/test/'))
)
```

For best performance, use a single-triple pattern on the THEN side of the rule. If a rule has multiple triple patterns on the THEN side, you can easily break it into multiple rules, each with a single-triple pattern, on the THEN side.

A **rulebase** is an object that contains rules. The following Oracle-supplied rulebases are provided:

- RDFS
- RDF (a subset of RDFS)
- OWLSIF (empty)
- RDFS++ (empty)
- OWL2EL (empty)
- OWL2RL (empty)
- OWLPrime (empty)
- SKOSCORE (empty)

The RDFS and RDF rulebases are created when you call the SEM_APIS.CREATE_RDF_NETWORK procedure to add RDF support to the database. The RDFS rulebase implements the RDFS inference rules, as described in the World Wide Web Consortium (W3C) *RDF Semantics* document at `http://www.w3.org/TR/rdf-mt/`. The RDF rulebase represents the RDF inference rules, which are a subset of the RDFS entailment rules. You can see the contents of these rulebases by examining the SEMR_RDFS and SEMR_RDF views.

You can also create user-defined rulebases using the SEM_APIS.CREATE_RULEBASE procedure. User-defined rulebases enable you to provide additional specialized inferencing capabilities.

For each rulebase, a table is created to hold rules in the rulebase, along with a view with a name in the format SEMR_*rulebase-name* (for example, SEMR_FAMILY_RB for a rulebase named `FAMILY_RB`). You must use this view to insert, delete, and modify rules in the rulebase. Each SEMR_*rulebase-name* view has the columns shown in Table 1-6.

Table 1-6   SEMR_rulebase-name View Columns

Column Name	Data Type	Description
RULE_NAME	VARCHAR2(30)	Name of the rule
ANTECEDENTS	VARCHAR2(4000)	IF side pattern for the antecedents
FILTER	VARCHAR2(4000)	(Not supported.)
CONSEQUENTS	VARCHAR2(4000)	THEN side pattern for the consequents

Table 1-6 (Cont.) SEMR_rulebase-name View Columns

Column Name	Data Type	Description
ALIASES	SEM_ALIASES	One or more namespaces to be used. (The SEM_ALIASES data type is described in Using the SEM_MATCH Table Function to Query RDF data.)

Information about all rulebases is maintained in the SEM_RULEBASE_INFO view, which has the columns shown in Table 1-7 and one row for each rulebase.

Table 1-7 SEM_RULEBASE_INFO View Columns

Column Name	Data Type	Description
OWNER	VARCHAR2(30)	Owner of the rulebase
RULEBASE_NAME	VARCHAR2(25)	Name of the rulebase
RULEBASE_VIEW_NAME	VARCHAR2(30)	Name of the view that you must use for any SQL statements that insert, delete, or modify rules in the rulebase
STATUS	VARCHAR2(30)	Contains `VALID` if the rulebase is valid, `INPROGRESS` if the rulebase is being created, or `FAILED` if a system failure occurred during the creation of the rulebase.

Example 1-3 Inserting a Rule into a Rulebase

Example 1-3 creates a rulebase named `family_rb`, and then inserts a rule named `grandparent_rule` into the `family_rb` rulebase. This rule says that if a person is the parent of a child who is the parent of a child, that person is a grandparent to (that is, has the `grandParentOf` relationship with respect to) the child's child. It also specifies a namespace to be used. (This example is an excerpt from Example 1-130 in Example: Family Information.)

```
EXECUTE SEM_APIS.CREATE_RULEBASE('family_rb', network_owner=>'RDFUSER',
network_name=>'NET1');

INSERT INTO rdfuser.net1#semr_family_rb VALUES(
  'grandparent_rule',
  '(?x :parentOf ?y) (?y :parentOf ?z)',
  NULL,
  '(?x :grandParentOf ?z)',
  SEM_ALIASES(SEM_ALIAS('','http://www.example.org/family/')));
```

Note that the kind of grandparent rule shown in Example 1-3 can be implemented using the OWL 2 property chain construct. For information about property chain handling, see Property Chain Handling.

Example 1-4 Using Rulebases for Inferencing

You can specify one or more rulebases when calling the SEM_MATCH table function (described in Using the SEM_MATCH Table Function to Query RDF data), to control the behavior of queries against RDF data. Example 1-4 refers to the `family_rb` rulebase and to the `grandParentOf` relationship created in Example 1-3, to find all grandfathers (grandparents who are male) and their grandchildren. (This example is an excerpt from Example 1-130 in Example: Family Information.)

```
-- Select all grandfathers and their grandchildren from the family RDF graph.
-- Use inferencing from both the RDFS and family_rb rulebases.
SELECT x$rdfterm grandfather, y$rdfterm grandchild
```

```
FROM TABLE(SEM_MATCH(
  'PREFIX  rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
   PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
   PREFIX     : <http://www.example.org/family/>
   SELECT ?x ?y
   WHERE {?x :grandParentOf ?y . ?x rdf:type :Male}',
  SEM_Models('family'),
  SEM_Rulebases('RDFS','family_rb'),
  null,  null, null,
  ' PLUS_RDFT=VC ',
  null, null,
  'RDFUSER', 'NET1'));
```

For information about support for native OWL inferencing, see Using OWL Inferencing.

1.4.8 Inferred Graphs

An **inferred graph** is an object containing precomputed triples that can be inferred from applying a specified set of rulebases to a specified set of RDF graphs. If a SEM_MATCH query refers to any rulebases, an inferred graph must exist for each rulebase-RDF graph combination in the query.

> **Note:**
>
> **Inferred graphs** were called as **Entailments** in the previous book versions (prior to Oracle Database Release 23ai). See Changes in Terminology and Subprograms for more information.

To create an inferred graph, use the SEM_APIS.CREATE_INFERRED_GRAPH procedure. To drop (delete) an inferred graph, use the SEM_APIS.DROP_INFERRED_GRAPH procedure.

When you create an inferred graph, a view for the triples associated with the inferred graph is also created under the network owner's schema. This view has a name in the format SEMI_*inferred-graph-name*, and it is visible only to the owner of the inferred graph and to users with suitable privileges. Each SEMI_*inferred-graph-name* view contains a row for each triple (stored as a link in a network), and it has the same columns as the SEMM_*rdf-graph-name* view, which is described in Table 1-4 in Metadata for Models.

Information about all inferred graphs is maintained in the SEM_RULES_INDEX_INFO view, which has the columns shown in Table 1-8 and one row for each inferred graph.

Table 1-8   SEM_RULES_INDEX_INFO View Columns

Column Name	Data Type	Description
OWNER	VARCHAR2(30)	Owner of the inferred graph.
INDEX_NAME	VARCHAR2(25)	Name of the inferred graph.
INDEX_VIEW_NAME	VARCHAR2(30)	Name of the view that you must use for any SQL statements that insert, delete, or modify rules in the inferred graph.
STATUS	VARCHAR2(30)	Contains `VALID` if the inferred graph is valid, `INVALID` if the inferred graph is not valid, `INCOMPLETE` if the inferred graph is incomplete (similar to `INVALID` but requiring less time to re-create), `INPROGRESS` if the inferred graph is being created, or `FAILED` if a system failure occurred during the creation of the inferred graph.

Table 1-8 (Cont.) SEM_RULES_INDEX_INFO View Columns

Column Name	Data Type	Description
MODEL_COUNT	NUMBER	Number of RDF graphs included in the inferred graph.
RULEBASE_COUNT	NUMBER	Number of rulebases included in the inferred graph.

Information about all database objects, such as RDF graphs and rulebases, related to inferred graphs is maintained in the SEM_RULES_INDEX_DATASETS view. This view has the columns shown in Table 1-9 and one row for each unique combination of values of all the columns.

Table 1-9 SEM_RULES_INDEX_DATASETS View Columns

Column Name	Data Type	Description
INDEX_NAME	VARCHAR2(25)	Name of the inferred graph
DATA_TYPE	VARCHAR2(8)	Type of data included in the inferred graph. Examples: `MODEL` for an RDF graph and `RULEBASE` for a rulebase
DATA_NAME	VARCHAR2(25)	Name of the object of the type in the DATA_TYPE column

Example 1-5 creates an inferred graph named `family_rb_rix_family`, using the `family` graph and the `RDFS` and `family_rb` rulebases. (This example is an excerpt from Example 1-130 in Example: Family Information.)

Example 1-5 Creating an inferred graph

```
BEGIN
  SEM_APIS.CREATE_INFERRED_GRAPH(
    'rdfs_rix_family',
    sem_models('family'),
    sem_Rulebases('RDFS','family_rb'),
    network_owner=>'RDFUSER', network_name=>'NET1');
END;
/
```

1.4.9 RDF Graph Collections

An RDF graph collection is a logical graph that can be used in a SEM_MATCH query. An RDF graph collection is the result of a UNION or UNION ALL operation on one or more RDF graphs and/or inferred graphs. Using RDF graph collections can help simplify the development process. However, for operational workloads in production, it is recommended you use single RDF graphs where possible.

> **Note:**
>
> **RDF Graph Collections** were called as **Virtual Models** in the previous book versions (prior to Oracle Database Release 23ai). See Changes in Terminology and Subprograms for more information.

Queries against a single RDF graph can more effectively use partition pruning and are able to use local optimizer statistics for the single-graph's partition, compared to queries against an

RDF graph collection. Queries against an RDF graph collection use global optimizer statistics for the entire RDF network, which can be less accurate than local, graph-level statistics. Hence, where possible you must combine the datasets that are queried together into a single RDF graph.

Besides using RDF graph collections for development, you can also use them if you need to access data across multiple RDF graphs in a single query, and also need to keep the individual RDF graphs separate for other queries. However, if possible, you must combine the datasets that are queried together into a single RDF graph.

Using an RDF graph collection, during the development phase of a project, provides the following benefits:

- It can simplify management of access privileges for RDF data. For example, assume that you have created three RDF graphs and one inferred graph based on the three graphs and the OWLPrime rulebase. Without an RDF graph collection, you must individually grant and revoke access privileges for each RDf graph and the inferred graph. However, if you create an RDF graph collection that contains the three RDF graphs and the inferred graph, you will only need to grant and revoke access privileges for the single RDF graph collection.
- It can facilitate rapid updates to RDF graphs. For example, assume that RDF graph collection VM1 contains RDF graph M1 and inferred graph R1 (that is, VM1 = M1 UNION ALL R1), and assume that RDF graph M1_UPD is a copy of M1 that has been updated with additional triples and that R1_UPD is an inferred graph created for M1_UPD. Now, to have user queries over VM1 go to the updated RDF graph and inferred graph, you can redefine RDF graph collection VM1 (that is, VM1 = M1_UPD UNION ALL R1_UPD).
- It can simplify query specification because querying an RDF graph collection is equivalent to querying multiple RDF graphs in a SEM_MATCH query. For example, assume that RDF graphs m1, m2, and m3 already exist, and that an inferred graph has been created for m1, m2 ,and m3 using the OWLPrime rulebase. You could create an RDF graph collection vm1 as follows:

  ```
  EXECUTE sem_apis.create_rdf_graph_collection('vm1', sem_models('m1', 'm2', 'm3'),
                                        sem_rulebases('OWLPRIME'),
                                        network_owner=>'RDFUSER',
                                        network_name=>'NET1');
  ```

  To query the RDF graph collection, use the RDF graph collection name as if it were a RDF graph in a SEM_MATCH query. For example, the following query on the RDF graph collection:

  ```
  SELECT * FROM TABLE (sem_match('{…}', sem_models('vm1'), null, …));
  ```

  is equivalent to the following query on all the individual RDF graphs:

  ```
  SELECT * FROM TABLE (sem_match('{…}', sem_models('m1', 'm2', 'm3'),
                                        sem_rulebases('OWLPRIME'), …));
  ```

  A SEM_MATCH query over an RDF graph collection will query either the *SEMV* or *SEMU* view (*SEMU* by default and *SEMV* if the 'ALLOW_DUP=T' option is specified) rather than querying the UNION or UNION ALL of each RDF graph and inferred graph. For information about these views and options, see the reference section for the SEM_APIS.CREATE_RDF_GRAPH_COLLECTION procedure.

RDF graph collections use views (described later in this section) and add some metadata entries, but do not significantly increase system storage requirements.

To create an RDF graph collection, use the SEM_APIS.CREATE_RDF_GRAPH_COLLECTION procedure. To drop (delete) an RDF graph collection, use the SEM_APIS.DROP_RDF_GRAPH_COLLECTION procedure. an RDF graph collection is dropped automatically if any of its component RDF graphs, rulebases, or inferred graph are dropped. To replace an RDF graph collection without dropping it, use the SEM_APIS.CREATE_RDF_GRAPH_COLLECTION procedure with the `REPLACE=T` option. Replacing an RDF graph collection allows you to redefine it while maintaining any access privileges.

To query an RDF graph collection, specify the RDF graph collection name in the `models` parameter of the SEM_MATCH table function, as shown in Example 1-6.

For information about the SEM_MATCH table function, see Using the SEM_MATCH Table Function to Query RDF data, which includes information using certain attributes when querying an RDF graph collection.

When you create an RDF graph collection, an entry is created for it in the SEM_MODEL$ view, which is described in Table 1-3 in RDF Graphs. However, the values in several of the columns are different for RDF graph collections as opposed to RDF graphs, as explained in Table 1-10.

Table 1-10 SEM_MODEL$ View Column Explanations for RDF graph collections

Column Name	Data Type	Description
OWNER	VARCHAR2(30)	Schema of the owner of the RDF graph collection
MODEL_ID	NUMBER	Unique model ID number, automatically generated. Will be a negative number, to indicate that this is an RDF graph collection.
MODEL_NAME	VARCHAR2(25)	Name of the RDF graph collection
TABLE_NAME	VARCHAR2(30)	Null for an RDF graph collection
COLUMN_NAME	VARCHAR2(30)	Null for an RDF graph collection
MODEL_TABLESPACE_NAME	VARCHAR2(30)	Null for an RDF graph collection

Information about all RDF graph collections is maintained in the SEM_VMODEL_INFO view, which has the columns shown in Table 1-11 and one row for each RDF graph collection.

Table 1-11 SEM_VMODEL_INFO View Columns

Column Name	Data Type	Description
OWNER	VARCHAR2(30)	Owner of the RDF graph collection
VIRTUAL_MODEL_NAME	VARCHAR2(25)	Name of the RDF graph collection
UNIQUE_VIEW_NAME	VARCHAR2(30)	Name of the view that contains unique triples in the RDF graph collection, or null if the view was not created
DUPLICATE_VIEW_NAME	VARCHAR2(30)	Name of the view that contains duplicate triples (if any) in the RDF graph collection

Table 1-11 (Cont.) SEM_VMODEL_INFO View Columns

Column Name	Data Type	Description
STATUS	VARCHAR2(30)	Contains `VALID` if the associated inferred graph is valid, `INVALID` if the inferred graph is not valid, `INCOMPLETE` if the inferred graph is incomplete (similar to `INVALID` but requiring less time to re-create), `INPROGRESS` if the inferred graph is being created, `FAILED` if a system failure occurred during the creation of the inferred graph, or `NORIDX` if no inferred graph is associated with the RDF graph collection. In the case of multiple inferred graphs, the lowest status among all of the component inferred graphs is used as the RDF graph collection's status (`INVALID` < `INCOMPLETE` < `VALID`).
MODEL_COUNT	NUMBER	Number of RDF graphs in the RDF graph collection
RULEBASE_COUNT	NUMBER	Number of rulebases used for the RDF graph collection
RULES_INDEX_COUNT	NUMBER	Number of inferred graphs in the RDF graph collection

Information about all objects (RDF graphs, rulebases, and inferred graphs) related to RDF graph collections is maintained in the SEM_VMODEL_DATASETS view. This view has the columns shown in Table 1-12 and one row for each unique combination of values of all the columns.

Table 1-12 SEM_VMODEL_DATASETS View Columns

Column Name	Data Type	Description
VIRTUAL_MODEL_NAME	VARCHAR2(25)	Name of the RDF graph collection
DATA_TYPE	VARCHAR2(8)	Type of object included in the RDF graph collection. Examples: `MODEL` for an RDF graph, `RULEBASE` for a rulebase, or `RULEIDX` for an inferred graph
DATA_NAME	VARCHAR2(25)	Name of the object of the type in the DATA_TYPE column

Example 1-6 Querying an RDF Graph Collection

```
SELECT COUNT(protein)
  FROM TABLE (SEM_MATCH (
    'SELECT ?protein
     WHERE {
      ?protein rdf:type :Protein .
      ?protein :citation ?citation .
      ?citation :author "Bairoch A."}',
    RDF_MODELS('UNIPROT_VM'),
    NULL,
    SEM_ALIASES(SEM_ALIAS('', 'http://purl.uniprot.org/core/')),
    NULL,
    NULL,
    'ALLOW_DUP=T',
    NULL,
    NULL,
    'RDFUSER','NET1'));
```

1.4.10 Named Graphs

RDF Graph supports the use of named graphs, which are described in the "RDF Dataset" section of the W3C *SPARQL Query Language for RDF* recommendation (`http://www.w3.org/TR/rdf-sparql-query/#rdfDataset`).

This support is provided by extending an RDF triple consisting of the traditional subject, predicate, and object, to include an additional component to represent a **graph name**. The extended RDF triple, despite having four components, will continue to be referred to as an *RDF triple* in this document. In addition, the following terms are sometimes used:

- **N-Triple** is a format that does not allow extended triples. Thus, n-triples can include only triples with three components.
- **N-Quad** is a format that allows both "regular" triples (three components) and extended triples (four components, including the graph name). For more information, see `http://www.w3.org/TR/2013/NOTE-n-quads-20130409/`.

  To load a file containing extended triples (possibly mixed with regular triples) into an Oracle database, the input file must be in N-Quad format.

The graph name component of an RDF triple must either be null or a URI. If it is null, the RDF triple is said to belong to a **default graph**; otherwise it is said to belong to a named graph whose name is designated by the URI.

Additionally, to support named graphs in SDO_RDF_TRIPLE_S object type (described in Semantic Data Types_ Constructors_ and Methods), a new syntax is provided for specifying a model-graph, that is, a combination of model (RDF graph) and named graph (if any) together, and the RDF_M_ID attribute holds the identifier for a model-graph: a combination of model (RDF graph) ID and value ID for the named graph (if any). The name of a model-graph is specified as *rdf_graph_name*, and if a named graph is present, followed by the colon (`:`) separator character and the name of the named graph (which must be a URI and enclosed within angle brackets `< >`).

For example, in a medical data set the named graph component for each RDF triple might be a URI based on patient identifier, so there could be as many named graphs as there are unique patients, with each named graph consisting of data for a specific patient.

For information about performing specific operations with named graphs, see the following:

- Using constructors and methods: Semantic Data Types_ Constructors_ and Methods
- Loading: Loading N-Quad Format Data into a Staging Table Using an External Table and Loading Data into Named Graphs Using INSERT Statements
- Querying: GRAPH Keyword Support and Expressions in the SELECT Clause
- Inferencing: Using Named Graph Based Inferencing (Global and Local)
- Data Formats Related to Named Graph Support

1.4.10.1 Data Formats Related to Named Graph Support

TriG and N-QUADS are two popular data formats that provide graph names (or context) to triple data. The graph names (context) can be used in a variety of different ways. Typical usage includes, but is not limited to, the grouping of triples for ease of management, localized query, localized inference, and provenance.

Example 1-7 RDF Data Encoded in TriG Format

Example 1-7 shows an RDF data set encoded in TriG format. It contains a default graph and a named graph.

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .

# Default graph
{
  <http://my.com/John> dc:publisher <http://publisher/Xyz> .
}

# A named graph
<http://my.com/John> {
  <http://my.com/John> foaf:name "John Doe" .
}
```

When loading the TriG file from Example 1-7 into a `DatasetGraphOracleSem` object (for example, using Example 7-12 in Bulk Loading Using RDF Semantic Graph Support for Apache Jena, but replacing the constant `"N-QUADS"` with `"TRIG"`), the triples in the default graph will be loaded into Oracle Database as triples with null graph names, and the triples in the named graphs will be loaded into Oracle Database with the designated graph names.

Example 1-8 N-QUADS Format Representation

N-QUADS format is a simple extension of the existing N-TRIPLES format by adding an optional fourth column (graph name or context). Example 1-8 shows the N-QUADS format representation of the TriG file from Example 1-7.

```
<http://my.com/John> <http://purl.org/dc/elements/1.1/publisher> <http://publisher/Xyz> .
<http://my.com/John> <http://xmlns.com/foaf/0.1/name> "John Doe" <http://my.com/John>
```

When loading an N-QUADS file into a `DatasetGraphOracleSem` object (see Example 7-12), lines without the fourth column will be loaded into Oracle Database as triples with null graph names, and lines with a fourth column will be loaded into Oracle Database with the designated graph names.

1.4.11 RDF Data Security Considerations

The following database security considerations apply to the use of RDF data:

- When an RDF graph or inferred graph is created, the owner gets the SELECT privilege with the GRANT option on the associated view. Users that have the SELECT privilege on these views can perform SEM_MATCH queries against the associated RDF graph or inferred graph.
- When a rulebase is created, the owner gets the SELECT, INSERT, UPDATE, and DELETE privileges on the rulebase, with the GRANT option. Users that have the SELECT privilege on a rulebase can create an inferred graph that includes the rulebase. The INSERT, UPDATE, and DELETE privileges control which users can modify the rulebase and how they can modify it.
- To perform data manipulation language (DML) operations on an RDF graph, a user must have DML privileges for the corresponding base table.
- The creator of the base table corresponding to an RDF graph can grant privileges to other users.
- To perform data manipulation language (DML) operations on a rulebase, a user must have the appropriate privileges on the corresponding database view.

- The creator of an RDF graph can grant SELECT privileges on the corresponding database view to other users.
- A user can query only those RDF graphs for which that user has SELECT privileges to the corresponding database views.
- Only the creator of an RDF graph or a rulebase can drop it.

1.4.12 RDF Privilege Considerations

The following database privilege-related considerations apply to the use of RDF networks:

- The network owner user whose schema will hold the tables and views for the RDF network must have the following roles and priviliges:
  `GRANT CONNECT, RESOURCE, CREATE VIEW TO <network_owner_user>;`
- The network owner requires quota on the tablespace that will contain the network.

1.5 RDF Metadata Tables and Views

Oracle Database maintains several tables and views in the network owner's schema to hold metadata related to RDF data.

Some of these tables and views are created by the SEM_APIS.CREATE_RDF_NETWORK procedure, as explained in Quick Start for Using Semantic Data, and some are created only as needed.Table 1-13 lists the tables and views in alphabetical order. (In addition, several tables and views are created for Oracle internal use, and these are accessible only by network owners of the schema-private RDF networks).

Table 1-13 RDF Metadata Tables and Views

Name	Contains Information About	Described In
RDF_CRS_URI$	Available EPSG spatial reference system URIs	Spatial Support
RDF_VALUE$	Subjects, properties, and objects used to represent statements	Statements
SEM_DTYPE_INDEX_INFO	All data type indexes in the network	Using Data Type Indexes
SEM_MODEL$	All RDF graphs defined in the database	RDF Graphs
SEM_NETWORK_INDEX_INFO$	RDF network indexes	SEM_NETWORK_INDEX_INFO View
SEM_RULEBASE_INFO	Rulebases	Inferencing: Rules and Rulebases
SEM_RULES_INDEX_DATASETS	Database objects used in inferred graphs	Inferred Graphs
SEM_RULES_INDEX_INFO	Inferred graphs	Inferred Graphs
SEM_VMODEL_INFO	RDF graph collections	RDF Graph Collections
SEM_VMODEL_DATASETS	Database objects used in RDF graph collections	RDF Graph Collections
SEMCL_*inferred-graph-name*	`owl:sameAs` clique members and canonical representatives	Optimizing owl:sameAs Inference

Table 1-13 (Cont.) RDF Metadata Tables and Views

Name	Contains Information About	Described In
SEMI_*inferred-graph-name*	Triples in the specified inferred graph	Inferred Graphs
SEMM_*rdf-graph-name*	Triples in the specified RDF graph	RDF Graphs
SEMR_*rulebase-name*	Rules in the specified rulebase	Inferencing: Rules and Rulebases
SEMU_*rdf-collection-name*	Unique triples in the RDF graph collection	RDF Graph Collections
SEMV_*rdf-collection-name*	Triples in the RDF graph collection	RDF Graph Collections

1.6 RDF Data Types, Constructors, and Methods

The SDO_RDF_TRIPLE_S object type is used for representing the edges (that is, triples and quads) of RDF graphs.

The SDO_RDF_TRIPLE_S object type (the _S for storage) stores persistent RDF data in the database.

The SDO_RDF_TRIPLE_S type has references to the data, because the actual RDF data is stored only in the central RDF schema. This type has methods to retrieve the entire triple or part of the triple.

Note:

Blank nodes are always reused within an RDF graph and cannot be reused across graphs.

The SDO_RDF_TRIPLE_S type is used to store the triples in database tables.

The SDO_RDF_TRIPLE_S object type has the following attributes:

```
SDO_RDF_TRIPLE_S (
  RDF_C_ID NUMBER, -- Canonical object value ID
  RDF_M_ID NUMBER, -- Model (or Model-Graph) ID
  RDF_S_ID NUMBER, -- Subject value ID
  RDF_P_ID NUMBER, -- Property value ID
  RDF_O_ID NUMBER) -- Object value ID
```

The SDO_RDF_TRIPLE_S type has the following methods that retrieve the name of the RDF graph (or model-graph), or a part (subject, property, or object) of a triple:

```
GET_MODEL(
 NETWORK_OWNER VARCHAR2 DEFAULT NULL,
 NETWORK_NAME  VARCHAR2 DEFAULT NULL) RETURNS VARCHAR2
GET_SUBJECT(
 NETWORK_OWNER VARCHAR2 DEFAULT NULL,
 NETWORK_NAME  VARCHAR2 DEFAULT NULL) RETURNS VARCHAR2
GET_PROPERTY(
 NETWORK_OWNER VARCHAR2 DEFAULT NULL,
 NETWORK_NAME  VARCHAR2 DEFAULT NULL) RETURNS VARCHAR2
```

```
GET_OBJECT(
 NETWORK_OWNER VARCHAR2 DEFAULT NULL,
 NETWORK_NAME  VARCHAR2 DEFAULT NULL) RETURNS CLOB
GET_OBJ_VALUE(
 NETWORK_OWNER VARCHAR2 DEFAULT NULL,
 NETWORK_NAME VARCHAR2 DEFAULT NULL) RETURNS VARCHAR2
```

Example 1-9 shows some of the SDO_RDF_TRIPLE_S methods.

Example 1-9   SDO_RDF_TRIPLE_S Methods

```
-- Find all articles that reference Article2.
SELECT a.triple.GET_SUBJECT('RDFUSER','NET1') AS subject
    FROM RDFUSER.NET1#RDFT_ARTICLES a
    WHERE a.triple.GET_PROPERTY('RDFUSER','NET1') = '<http://purl.org/dc/terms/
references>'
    AND a.triple.GET_OBJ_VALUE('RDFUSER','NET1') = '<http://nature.example.com/
Article2>';

SUBJECT
--------------------------------------------------------------------------------
<http://nature.example.com/Article1>

-- Find all triples with Article1 as subject.
SELECT a.triple.GET_SUBJECT('RDFUSER','NET1') AS subject,
       a.triple.GET_PROPERTY('RDFUSER','NET1') AS property,
       a.triple.GET_OBJ_VALUE('RDFUSER','NET1') AS object
    FROM RDFUSER.NET1#RDFT_ARTICLES a
    WHERE a.triple.GET_SUBJECT('RDFUSER','NET1') = '<http://nature.example.com/
Article1>';

SUBJECT
--------------------------------------------------------------------------------
PROPERTY
--------------------------------------------------------------------------------
OBJECT
--------------------------------------------------------------------------------
<http://nature.example.com/Article1>
<http://purl.org/dc/elements/1.1/title>
"All about XYZ"

<http://nature.example.com/Article1>
<http://purl.org/dc/elements/1.1/creator>
"Jane Smith"

<http://nature.example.com/Article1>
<http://purl.org/dc/terms/references>
<http://nature.example.com/Article2>

<http://nature.example.com/Article1>
<http://purl.org/dc/terms/references>
<http://nature.example.com/Article3

-- Find all objects where the subject is Article1.
SELECT a.triple.GET_OBJ_VALUE('RDFUSER','NET1') AS object
    FROM RDFUSER.NET1#RDFT_ARTICLES a
    WHERE a.triple.GET_SUBJECT('RDFUSER','NET1') = '<http://nature.example.com/
Article1>';

OBJECT
--------------------------------------------------------------------------------
"All about XYZ"
"Jane Smith"
```

```
<http://nature.example.com/Article2>
<http://nature.example.com/Article3>

-- Find all triples where Jane Smith is the object.
SELECT a.triple.GET_SUBJECT('RDFUSER','NET1') AS subject,
       a.triple.GET_PROPERTY('RDFUSER','NET1') AS property,
       a.triple.GET_OBJ_VALUE('RDFUSER','NET1') AS object
    FROM RDFUSER.NET1#RDFT_ARTICLES a
    WHERE a.triple.GET_OBJ_VALUE('RDFUSER','NET1') = '"Jane Smith"';

SUBJECT
--------------------------------------------------------------------------------
PROPERTY
--------------------------------------------------------------------------------
OBJECT
--------------------------------------------------------------------------------
<http://nature.example.com/Article1>
<http://purl.org/dc/elements/1.1/creator>
"Jane Smith"
```

- Constructors for Inserting Triples

1.6.1 Constructors for Inserting Triples

The following constructor formats are available for inserting triples into a model table. The only difference is that in the second format the data type for the object is CLOB, to accommodate very long literals.

```
SDO_RDF_TRIPLE_S (
  model_name    VARCHAR2, -- Model name
  subject       VARCHAR2, -- Subject
  property      VARCHAR2, -- Property
  object        VARCHAR2, -- Object
  network_owner VARCHAR2 DEFAULT NULL,
  network_name  VARCHAR2 DEFAULT NULL)
  RETURN     SELF;

SDO_RDF_TRIPLE_S (
  model_name    VARCHAR2, -- Model name
  subject       VARCHAR2, -- Subject
  property      VARCHAR2, -- Property
  object        CLOB,     -- Object
  network_owner VARCHAR2 DEFAULT NULL,
  network_name  VARCHAR2 DEFAULT NULL)
  RETURN SELF;
```

Example 1-10 uses the first constructor format to insert several triples.

Example 1-10 SDO_RDF_TRIPLE_S Constructor to Insert Triples

```
INSERT INTO RDFUSER.NET1#RDFT_ARTICLES VALUES (
  SDO_RDF_TRIPLE_S ('articles','<http://nature.example.com/Article1>',
    '<http://purl.org/dc/elements/1.1/creator>',
    '"Jane Smith"',
    'RDFUSER',
    'NET1'));

INSERT INTO RDFUSER.NET1#RDFT_ARTICLES VALUES (
  SDO_RDF_TRIPLE_S ('articles:<http://examples.com/ns#Graph1>',
    '<http://nature.example.com/Article102>',
    '<http://purl.org/dc/elements/1.1/creator>',
    '_:b1',
```

```
    'RDFUSER',
    'NET1'));

INSERT INTO RDFUSER.NET1#RDFT_ARTICLES VALUES (
  SDO_RDF_TRIPLE_S ('articles:<http://examples.com/ns#Graph1>',
    '_:b2',
    '<http://purl.org/dc/elements/1.1/creator>',
    '_:b1',
    'RDFUSER',
    'NET1'));
```

1.7 Using the SEM_MATCH Table Function to Query RDF Data

To query RDF data, use the SEM_MATCH table function.

> **Note:**
>
> The SEM_MATCH table function is supported only if Oracle JVM is enabled on your Oracle Autonomous Database Serverless deployments. To enable Oracle JVM, see Use Oracle Java in *Using Oracle Autonomous Database Serverless* for more information.

This function has the following attributes:

```
SEM_MATCH(
  query         VARCHAR2,
  models        SEM_MODELS,
  rulebases     SEM_RULEBASES,
  aliases       SEM_ALIASES,
  filter        VARCHAR2,
  index_status  VARCHAR2     DEFAULT NULL,
  options       VARCHAR2     DEFAULT NULL,
  graphs        SEM_GRAPHS   DEFAULT NULL,
  named_graphs  SEM_GRAPHS   DEFAULT NULL,
  network_owner VARCHAR2     DEFAULT NULL,
  network_name  VARCHAR2     DEFAULT NULL
 ) RETURN ANYDATASET;
```

The `query` and `models` attributes are required. The other attributes are optional (that is, each can be a null value).

The `query` attribute is a string literal (or concatenation of string literals) with one or more triple patterns, usually containing variables. (The `query` attribute cannot be a bind variable or an expression involving a bind variable.) A triple pattern is a triple of atoms followed by a period. Each atom can be a variable (for example, `?x`), a qualified name (for example, `rdf:type`) that is expanded based on the default namespaces and the value of the aliases attribute, or a full URI (for example, `<http://www.example.org/family/Male>`). In addition, the third atom can be a numeric literal (for example, `3.14`), a plain literal (for example, `"Herman"`), a language-tagged plain literal (for example, `"Herman"@en`), or a typed literal (for example, `"123"^^xsd:int`).

For example, the following `query` attribute specifies three triple patterns to find grandfathers (that is, grandparents who are also male) and the height of each of their grandchildren:

```
'SELECT * WHERE { ?x :grandParentOf ?y . ?x rdf:type :Male . ?y :height ?h }'
```

The `models` attribute identifies the RDF graphs to use. Its data type is SEM_MODELS, which has the following definition: `TABLE OF VARCHAR2(25)`. If you are querying an RDF graph collection, specify only the name of the RDF graph collection and no other RDF graphs. (RDF graph collections are explained in RDF Graph Collections.)

The `rulebases` attribute identifies one or more rulebases whose rules are to be applied to the query. Its data type is SEM_RULEBASES, which has the following definition: `TABLE OF VARCHAR2(25)`. If you are querying an RDF graph collection, this attribute must be null.

The `aliases` attribute identifies one or more namespaces, in addition to the default namespaces, to be used for expansion of qualified names in the query pattern. Its data type is SEM_ALIASES, which has the following definition: `TABLE OF SEM_ALIAS`, where each SEM_ALIAS element identifies a namespace ID and namespace value. The SEM_ALIAS data type has the following definition: `(namespace_id VARCHAR2(30), namespace_val VARCHAR2(4000))`

The following default namespaces (`namespace_id` and `namespace_val` attributes) are used by the SEM_MATCH table function and the SEM_CONTAINS and SEM_RELATED operators:

```
('ogc', 'http://www.opengis.net/ont/geosparql#')
('ogcf', 'http://www.opengis.net/def/function/geosparql/')
('ogcgml', 'http://www.opengis.net/ont/gml#')
('ogcsf', 'http://www.opengis.net/ont/sf#')
('orardf', 'http://xmlns.oracle.com/rdf/')
('orageo', 'http://xmlns.oracle.com/rdf/geo/')
('owl',    'http://www.w3.org/2002/07/owl#')
('rdf',    'http://www.w3.org/1999/02/22-rdf-syntax-ns#')
('rdfs',   'http://www.w3.org/2000/01/rdf-schema#')
('xsd',    'http://www.w3.org/2001/XMLSchema#')
```

You can override any of these defaults by specifying the `namespace_id` value and a different `namespace_val` value in the `aliases` attribute.

The `filter` attribute identifies any additional selection criteria. If this attribute is not null, it should be a string in the form of a `WHERE` clause without the `WHERE` keyword. For example: `'(h >= ''6'')'` to limit the result to cases where the height of the grandfather's grandchild is 6 or greater (using the example of triple patterns earlier in this section).

Note:

Instead of using the `filter` attribute, you are encouraged to use the FILTER keyword inside your query pattern whenever possible (as explained in Graph Patterns: Support for Curly Brace Syntax_ and OPTIONAL_ FILTER_ UNION_ and GRAPH Keywords). Using the FILTER keyword is likely to give better performance because of internal optimizations. The `filter` argument, however, can be useful if you require SQL constructs that cannot be expressed with the FILTER keyword.

The `index_status` attribute lets you query RDF data even when the relevant inferred graph does not have a valid status. (If you are querying an RDF graph collection, this attribute refers to the inferred graph associated with the RDF graph collection.) If this attribute is null, the query returns an error if the inferred graph does not have a valid status. If this attribute is not null, it must be the string `INCOMPLETE` or `INVALID`. For an explanation of query behavior with different `index_status` values, see Performing Queries with Incomplete or Invalid Entailments.

The `options` attribute identifies options that can affect the results of queries. Options are expressed as keyword-value pairs. The following options are supported:

- `ALL_AJ_HASH`, `ALL_AJ_MERGE`, and `ALL_BGP_NL` are global query optimizer hints that specify that all anti joins for NOT EXISTS and MINUS operations should use the specified join type.
- `ALL_BGP_HASH` and `ALL_BGP_NL` are global query optimizer hints that specify that all inter-BGP joins (for example. the join between the root BGP and an OPTIONAL BGP) should use the specified join type. (BGP stands for *basic graph pattern*. From the W3C SPARQL Query Language for RDF Recommendation: "SPARQL graph pattern matching is defined in terms of combining the results from matching basic graph patterns. A sequence of triple patterns interrupted by a filter comprises a single basic graph pattern. Any graph pattern terminates a basic graph pattern."

  The `BGP_JOIN(USE_NL)` and `BGP_JOIN(USE_HASH)` HINT0 query optimizer hints can be used to control the join type with finer granularity.

  Example 1-17 shows the ALL_BGP_HASH option used in a SEM_MATCH query.
- `AUTO_HINTS=T` automatically detects and generates `USE_HASH` hints for unselective SPARQL queries.
- `ALL_LINK_HASH` and `ALL_LINK_NL` are global query optimizer hints that specify the join type for all RDF_LINK$ joins (that is, all joins between triple patterns within a BGP). `ALL_LINK_HASH` and `ALL_LINK_NL` can also be used within a HINT0 query optimizer hint for finer granularity.
- `ALL_MAX_PP_DEPTH(n)` is a global query optimizer hint that sets the maximum depth to use when evaluating * and + property path operators. The default value is 10. The `MAX_PP_DEPTH(n)` HINT0 hint can be used to specify maximum depth with finer granularity.
- `ALL_NO_MERGE` is a global query optimizer hint that adds `NO_MERGE` to each subquery in the generated SQL for a SPARQL query. This hint is used to ensure that a selective subquery in a SPARQL query is not merged with the other parts of the SPARQL query.
- `ALL_ORDERED` is a global query optimizer hint that specifies that the triple patterns in each BGP in the query should be evaluated in order.

  Example 1-17 shows the ALL_ORDERED option used in a SEM_MATCH query.
- `ALL_USE_PP_HASH` and `ALL_USE_PP_NL` are global query optimizer hints that specify the join type to use when evaluating property path expressions. The `USE_PP_HASH` and `USE_PP_NL` HINT0 hints can be used for specifying join type with finer granularity.
- `ALLOW_DUP=T` generates an underlying SQL statement that performs a "union all" instead of a union of the RDF graphs and inferred data (if applicable). This option may introduce more rows (duplicate triples) in the result set, and you may need to adjust the application logic accordingly. If you do not specify this option, duplicate triples are automatically removed across all the RDF graphs and inferred data to maintain the set semantics of merged RDF graphs; however, removing duplicate triples increases query processing time. In general, specifying `'ALLOW_DUP=T'` improves performance significantly when multiple RDF graphs are involved in a SEM_MATCH query.

  If you are querying an RDF graph collection, specifying `ALLOW_DUP=T` causes the SEMV_*vm_name* view to be queried; otherwise, the SEMU_*vm_name* view is queried.
- `ALLOW_PP_DUP=T` allows duplicate results for + and * property path queries. Allowing duplicate results may return the first result rows faster.

- `AS_OF [SCN, <SCN_VALUE>]` , where <SCN_VALUE> is a valid system change number, indicates that Flashback Query should be used to query the state of the RDF network as of the specified SCN.
- `AS_OF [TIMESTAMP, <TIMESTAMP_VALUE>]` , where <TIMESTAMP_VALUE> is a valid timestamp string with format 'YYYY/MM/DD HH24:MI:SS.FF', indicates that Flashback Query should be used to query the state of the RDF network as of the specified timestamp.
- `CLOB_AGG_SUPPORT=T` enables support for CLOB values for the following aggregates: MIN, MAX, GROUP_CONCAT, SAMPLE. Note that enabling CLOB support incurs a significant performance penalty.
- `CLOB_EXP_SUPPORT=T` enables support for CLOB values for some built-in SPARQL functions. Note that enabling CLOB support incurs a significant performance penalty.
- `CONSTRUCT_STRICT=T` eliminates invalid RDF triples from the result of SPARQL CONSTRUCT or SPARQL DESCRIBE syntax queries. RDF triples with literals in the subject position or literals or blank nodes in the predicate position are considered invalid.
- `CONSTRUCT_UNIQUE=T` eliminates duplicate RDF triples from the result of SPARQL CONSTRUCT or SPARQL DESCRIBE syntax queries.
- `DISABLE_IM_VIRTUAL_COL` specifies that the query compiler should not use in-memory virtual columns.
- `DISABLE_MVIEW` specifies that the query compiler should not use materialized views.
- `DISABLE_NULL_EXPR_JOIN` specifies that the query compiler should assume that all SELECT expressions produce non-null output.
- `DISABLE_SAMEAS_BLOOM` specifies that the query compiler should not use a Bloom filter when `owl:sameAs` triples are joined. (For detailed information, see the explanation of Bloom filters in *Oracle Database SQL Tuning Guide*.)
- `DO_UNESCAPE=T` causes characters in the following return columns to be unescaped according to the W3C N-Triples specification (`http://www.w3.org/TR/rdf-testcases/#ntriples`): var, var$_PREFIX, var$_SUFFIX, var$RDFCLOB, var$RDFLTYP, var$RDFLANG, and var$RDFTERM.

  See also the reference information for SEM_APIS.ESCAPE_CLOB_TERM, SEM_APIS.ESCAPE_CLOB_VALUE, SEM_APIS.ESCAPE_RDF_TERM, SEM_APIS.ESCAPE_RDF_VALUE, SEM_APIS.UNESCAPE_CLOB_TERM, SEM_APIS.UNESCAPE_CLOB_VALUE, SEM_APIS.UNESCAPE_RDF_TERM, and SEM_APIS.UNESCAPE_RDF_VALUE.
- `FINAL_VALUE_HASH` and `FINAL_VALUE_NL` are global query optimizer hints that specify the join method that should be used to obtain the lexical values for any query variables that are not used in a FILTER clause.
- `GRAPH_MATCH_UNNAMED=T` allows unnamed triples (null `G_ID`) to be matched inside GRAPH clauses. That is, two triples will satisfy the graph join condition if their graphs are equal or if one or both of the graphs are null. This option may be useful when your dataset includes unnamed TBOX triples or unnamed entailed triples.
- `HINT0={<hint-string>}` (pronounced and written "hint" and the number zero) specifies one or more keywords with hints to influence the execution plan and results of queries. Conceptually, a graph pattern with *n* triple patterns and referring to *m* distinct variables results in an (*n*+*m*)-way join: *n*-way self-join of the target RDFgraphs and optionally the corresponding inferred graph, and then *m* joins with RDF_VALUE$ for looking up the values for the *m* variables. A hint specification affects the join order and join type used for the query execution.

The hint specification, <*hint-string*>, uses keywords, some of which have parameters consisting of a sequence or set of aliases, or references, for individual triple patterns and variables used in the query. Aliases for triple patterns are of the form t*i* where *i* refers to the 0-based ordinal numbers of triple patterns in the query. For example, the alias for the first triple pattern in a query is `t0`, the alias for the second one is `t1`, and so on. Aliases for the variables used in a query are simply the names of those variables. Thus, `?x` will be used in the hint specification as the alias for a variable `?x` used in the graph pattern.

Hints used for influencing query execution plans include LEADING(<*sequence of aliases*>), USE_NL(<*set of aliases*>), USE_HASH(<*set of aliases*>), and INDEX(<*alias*> <*index_name*>). These hints have the same format and basic meaning as hints in SQL statements, which are explained in *Oracle Database SQL Language Reference*.

Example 1-12 shows the HINT0 option used in a SEM_MATCH query.

- `HTTP_METHOD=POST_PAR` indicates that the HTTP POST method with URL-encoded parameters pass should be used for the SERVICE request. The default option for requests is the HTTP GET method. For more information about SPARQL protocol, see `http://www.w3.org/TR/2013/REC-sparql11-protocol-20130321/#protocol`.
- `INF_ONLY=T` queries only the entailed graph for the specified RDF graphs and rulebases.
- `OVERLOADED_NL=T` specifies that a procedural nested loop execution should be used to join with an overloaded SERVICE clause.
- `PLUS_RDFT=T` can be used with SPARQL SELECT syntax (see Expressions in the SELECT Clause) to additionally return a *var*$RDFTERM CLOB column for each projected query variable. The value for this column is equivalent to the result of SEM_APIS.COMPOSE_RDF_TERM(*var*, *var*$RDFVTYP, *var*$RDFLTYP, *var*$RDFLANG, *var*$RDFCLOB). When using this option, the return columns for each variable var will be *var*, *var*$RDFVID, *var*$_PREFIX, *var*$_SUFFIX, *var*$RDFVTYP, *var*$RDFCLOB, *var*$RDFLTYP, *var*$RDFLANG, and *var*$RDFTERM.
- `PLUS_RDFT=VC` can be used with SPARQL SELECT syntax (see Expressions in the SELECT Clause) to additionally return a var$RDFTERM VARCHAR2(`NETWORK_MAX_STRING_SIZE`) column for each projected query variable. The value for this column is equivalent to the result of SEM_APIS.COMPOSE_RDF_TERM(var, var$RDFVTYP, var$RDFLTYP, var$RDFLANG). When using this option, the return columns for each variable var will be var, var$RDFVID, var$_PREFIX, var$_SUFFIX, var$RDFVTYP, var$RDFCLOB, var$RDFLTYP, var$RDFLANG, and var$RDFTERM. Note that when your RDF network is using `NETWORK_STORAGE_FORM=UNESC`, special characters in var$RDFTERM are automatically escaped to form syntactically valid RDF values. This may cause the size of var$RDFTERM to exceed `NETWORK_MAX_STRING_SIZE` and hence an error will be raised in such cases. To avoid the error, you can use `PLUS_RDFT=T` to return a CLOB instead.
- `PROJ_EXACT_VALUES=T` disables canonicalization of values returned from functions and of constant values used in value assignment statements. Such values are canonicalized by default.
- `SERVICE_CLOB=F` sets the column values of var$RDFCLOB to null instead of saving values when calling the service. If CLOB data is not needed in your application, performance can be improved by using this option to skip CLOB processing.
- `SERVICE_ESCAPE=F` disables character escaping for RDF literal values returned by SPARQL SERVICE calls. RDF literal values are escaped by default. If character escaping is not relevant for your application, performance can be improved by disabling character escaping.

- `SERVICE_JPDWN=T` is a query optimizer hint for using nested loop join in SPARQL SERVICE. Example 1-73 shows the `SERVICE_JPDWN=T` option used in a SEM_MATCH query.
- `SERVICE_PROXY=<proxy-string>` sets a proxy address to be used when performing http connections. The given proxy-string will be used in SERVICE queries. Example 1-76 shows a SEM_MATCH query including a proxy address.
- `STRICT_AGG_CARD=T` uses SPARQL semantics (one null row) instead of SQL semantics (zero rows) for aggregate queries with graph patterns that fail to match. This option incurs a slight performance penalty.
- `STRICT_DEFAULT=T` restricts the default graph to unnamed triples when no dataset information is specified.

The `graphs` attribute specifies the set of named graphs from which to construct the default graph for a SEM_MATCH query. Its data type is SEM_GRAPHS, which has the following definition: `TABLE OF VARCHAR2(4000)`. The default value for this attribute is `NULL`. When `graphs` is `NULL`, the "union all" of all default graphs in the set of RDF graphs specified in the `models` attribute is used as the default graph.

The `named_graphs` attribute specifies the set of named graphs that can be matched by a GRAPH clause. Its data type is SEM_GRAPHS, which has the following definition: `TABLE OF VARCHAR2(4000)`. The default value for this attribute is `NULL`. When named_graphs is `NULL`, all named graphs in the set of RDF graphs specified in the `models` attribute can be matched by a GRAPH clause.

The `network_owner` attribute specifies the schema that owns the RDF network that contains the RDF graph or RDF graph collection specified in the models attribute. This attribute should be non-null to query a schema-private RDF network.

The `network_name` attribute specifies the name of the RDF network that contains the RDF graph or graph collection specified in the models attribute. This attribute should be non-null to query a schema-private RDF network.

The SEM_MATCH table function returns an object of type ANYDATASET, with elements that depend on the input variables. In the following explanations, *var* represents the name of a variable used in the query. For each variable *var*, the result elements have the following attributes: *var*, *var*$RDFVID, *var*$_PREFIX, *var*$_SUFFIX, *var*$RDFVTYP, *var*$RDFCLOB, *var*$RDFLTYP, and *var*$RDFLANG.

In such cases, *var* has the lexical value bound to the variable, *var*$RDFVID has the VALUE_ID of the value bound to the variable, *var*$_PREFIX and *var*$_SUFFIX are the *prefix* and *suffix* of the value bound to the variable, *var*$RDFVTYP indicates the type of value bound to the variable (`URI`, `LIT` [literal], or `BLN` [blank node]), *var*$RDFCLOB has the lexical value bound to the variable if the value is a long literal, *var*$RDFLTYP indicates the type of literal bound if a literal is bound, and *var*$RDFLANG has the language tag of the bound literal if a literal with language tag is bound. *var*$RDFCLOB is of type CLOB, while all other attributes are of type VARCHAR2.

For a literal value or a blank node, its prefix is the value itself and its suffix is null. For a URI value, its prefix is the left portion of the value up to and including the rightmost occurrence of any of the three characters / (slash), # (pound), or : (colon), and its suffix is the remaining portion of the value to the right. For example, the prefix and suffix for the URI value `http://www.example.org/family/grandParentOf` are `http://www.example.org/family/` and `grandParentOf`, respectively.

Along with columns for variable values, a SEM_MATCH query that uses SPARQL SELECT syntax returns one additional NUMBER column, SEM$ROWNUM, which can be used to ensure the correct result ordering for queries that involve a SPARQL ORDER BY clause.

A SEM_MATCH query that uses SPARQL ASK syntax returns the columns ASK, ASK$RDFVID, ASK$_PREFIX, ASK$_SUFFIX, ASK$RDFVTYP, ASK$RDFCLOB, ASK$RDFLTYP, ASK$RDFLANG, and SEM$ROWNUM. This is equivalent to a SPARQL SELECT syntax query that projects a single `?ask` variable.

A SEM_MATCH query that uses SPARQL CONSTRUCT or SPARQL DESCRIBE syntax returns columns that contain RDF triple data rather than query result bindings. Such queries return values for subject, predicate and object components. See Graph Patterns: Support for SPARQL CONSTRUCT Syntaxfor details.

To use the SEM_RELATED operator to query an OWL ontology, see Using Semantic Operators to Query Relational Data.

When you are querying multiple RDF graphs, or querying one or more RDF graphs and the corresponding inferred graph, consider using RDF graph collections (explained in RDF Graph Collections) because of the potential performance benefits.

Example 1-11 SEM_MATCH Table Function

Example 1-11 selects all grandfathers (grandparents who are male) and their grandchildren from the `family` RDF graph, using inferencing from both the `RDFS` and `family_rb` rulebases. (This example is an excerpt from Example 1-130 in Example: Family Information.)

```
SELECT x$rdfterm grandfather, y$rdfterm grandchild
  FROM TABLE(SEM_MATCH(
    'PREFIX  rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
     PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
     PREFIX     : <http://www.example.org/family/>
     SELECT ?x ?y
     WHERE {?x :grandParentOf ?y . ?x rdf:type :Male}',
    SEM_Models('family'),
    SEM_Rulebases('RDFS','family_rb'),
    null,  null, null,
    ' PLUS_RDFT=VC ',
    null, null,
    'RDFUSER', 'NET1'));
```

Example 1-12 HINT0 Option with SEM_MATCH Table Function

Example 1-12 is functionally the same as Example 1-11, but it adds the `HINT0` option.

```
SELECT x$rdfterm grandfather, y$rdfterm grandchild
  FROM TABLE(SEM_MATCH(
    'PREFIX  rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
     PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
     PREFIX     : <http://www.example.org/family/>
     SELECT ?x ?y
     WHERE {?x :grandParentOf ?y . ?x rdf:type :Male}',
    SEM_Models('family'),
    SEM_Rulebases('RDFS','family_rb'),
    null,  null, null,
    ' PLUS_RDFT=VC HINT0={LEADING(t0 t1) USE_NL(?x ?y)}',
    null, null,
    'RDFUSER', 'NET1'));
```

Example 1-13 DISABLE_SAMEAS_BLOOM Option with SEM_MATCH Table Function

Example 1-12 specifies that the query compiler should not use a Bloom filter when `owl:sameAs` triples are joined.

```
SELECT select s, o
FROM table(sem_match('{ # HINT0={LEADING(t1 t0) USE_HASH(t0 t1)}
```

```
 ?s owl:sameAs ?o. ?o owl:sameAs ?s}', sem_models('M1'), null,null,null,null,
  ' DISABLE_SAMEAS_BLOOM ')) order by 1,2;
```

Example 1-14 SEM_MATCH Table Function

Example 1-14 uses the Pathway/Genome `BioPax` ontology to get all chemical compound types that belong to both `Proteins` and `Complexes`:

```
SELECT t.r
  FROM TABLE (SEM_MATCH (
      'PREFIX : <http://www.biopax.org/release1/biopax-release1.owl>
       SELECT ?r
       WHERE {
        ?r rdfs:subClassOf :Proteins .
        ?r rdfs:subClassOf :Complexes}',
      SEM_Models ('BioPax'),
      SEM_Rulebases ('rdfs'),
      NULL, NULL, NULL, '', NULL, NULL,
      'RDFUER','NET1')) t;
```

As shown in Example 1-14, the search pattern for the SEM_MATCH table function is specified using SPARQL syntax where the variable starts with the question-mark character (`?`). In this example, the variable `?r` must match to the same term, and thus it must be a subclass of both `Proteins` and `Complexes`.

- Performing Queries with Incomplete or Invalid Inferred Graphs
- Graph Patterns: Support for Curly Brace Syntax, and OPTIONAL, FILTER, UNION, and GRAPH Keywords
- Graph Patterns: Support for SPARQL ASK Syntax
- Graph Patterns: Support for SPARQL CONSTRUCT Syntax
- Graph Patterns: Support for SPARQL DESCRIBE Syntax
- Graph Patterns: Support for SPARQL SELECT Syntax
- Graph Patterns: Support for SPARQL 1.1 Constructs
- Graph Patterns: Support for SPARQL 1.1 Federated Query
- Inline Query Optimizer Hints
- Full-Text Search
- Spatial Support
- Flashback Query Support
- Best Practices for Query Performance
- Special Considerations When Using SEM_MATCH

1.7.1 Performing Queries with Incomplete or Invalid Inferred Graphs

You can query RDF data even when the relevant inferred graph does not have a valid status if you specify the string value `INCOMPLETE` or `INVALID` for the `index_status` attribute of the SEM_MATCH table function. (The inferred graph status is stored in the STATUS column of the SEM_RULES_INDEX_INFO view, which is described in Inferred Graphs. The SEM_MATCH table function is described in Using the SEM_MATCH Table Function to Query RDF Data.)

The index_status attribute value affects the query behavior as follows:

- If the inferred graph has a valid status, the query behavior is not affected by the value of the `index_status` attribute.
- If you provide no value or specify a null value for `index_status`, the query returns an error if the inferred graph does not have a valid status.
- If you specify the string `INCOMPLETE` for the `index_status` attribute, the query is performed if the status of the inferred graph is incomplete or valid.
- If you specify the string `INVALID` for the `index_status` attribute, the query is performed regardless of the actual status of the inferred graph (invalid, incomplete, or valid).

However, the following considerations apply if the status of the inferred graph is incomplete or invalid:

- If the status is incomplete, the content of an inferred graph may be approximate, because some triples that are inferable (due to the recent insertions into the underlying RDF graphs) may not actually be present in the inferred graph, and therefore results returned by the query may be inaccurate.
- If the status is invalid, the content of the inferred graph may be approximate, because some triples that are no longer inferable (due to recent modifications to the underlying RDF graphs or rulebases, or both) may still be present in the inferred graph, and this may affect the accuracy of the result returned by the query. In addition to possible presence of triples that are no longer inferable, some inferable rows may not actually be present in the inferred graph.

1.7.2 Graph Patterns: Support for Curly Brace Syntax, and OPTIONAL, FILTER, UNION, and GRAPH Keywords

The SEM_MATCH table function accepts the syntax for the graph pattern in which a sequence of triple patterns is enclosed within curly braces. The period is usually required as a separator unless followed by the OPTIONAL, FILTER, UNION, or GRAPH keyword. With this syntax, you can do any combination of the following:

- Use the OPTIONAL construct to retrieve results even in the case of a partial match
- Use the FILTER construct to specify a filter expression in the graph pattern to restrict the solutions to a query
- Use the UNION construct to match one of multiple alternative graph patterns
- Use the GRAPH construct (explained in GRAPH Keyword Support) to scope graph pattern matching to a set of named graphs

In addition to arithmetic operators (+, -, *, /), Boolean operators and logical connectives (||, &&, !), and comparison operators (<, >, <=, >=, =, !=), several built-in functions are available for use in FILTER clauses. Table 1-14 lists built-in functions that you can use in the FILTER clause. In the Description column of Table 1-14, *x*, *y*, and *z* are arguments of the appropriate types.

Table 1-14 Built-in Functions Available for FILTER Clause

Function	Description
ABS(RDF term)	Returns the absolute value of `term`. If `term` is a non-numerical value, returns null.

Table 1-14 (Cont.) Built-in Functions Available for FILTER Clause

Function	Description
BNODE(literal) or BNODE()	Constructs a blank node that is distinct from all blank nodes in the dataset of the query, and those created by this function in other queries. The form with no arguments results in a distinct blank node in every call. The form with a simple literal results in distinct blank nodes for different simple literals, and the same blank node for calls with the same simple literal.
BOUND(variable)	BOUND(x) returns `true` if `x` is bound (that is, non-null) in the result, `false` otherwise.
CEIL(RDF term)	Returns the closest number with no fractional part which is not less than term. If term is a non-numerical value, returns null.
COALESCE(term list)	Returns the first element on the argument list that is evaluated without raising an error. Unbound variables raise an error if evaluated. Returns null if there are no valid elements in the term list.
CONCAT(term list)	Returns an `xsd:String` value resulting of the concatenation of the string values in the term list.
CONTAINS(literal, match)	Returns `true` if the string `match` is found anywhere in `literal`. It returns `false` otherwise.
DATATYPE(literal)	DATATYPE(x) returns a URI representing the datatype of `x`.
DAY(argument)	Returns an integer corresponding to the day part of argument. If the argument is not a `dateTime` or `date` data type, it returns a null value.
ENCODE_FOR_URI(literal)	Returns a string where the reserved characters in `literal` are escaped and converted to its percent-encode form.
EXISTS(pattern)	Returns `true` if the pattern matches the query data set, using the current bindings in the containing group graph pattern and the current active graph. If there are no matches, it returns `false`.
FLOOR(RDF term)	Returns the closest number with no fractional part which is less than `term`. If `term` is a non-numerical value, returns null.
HOURS(argument)	Returns an integer corresponding to the hours part of `argument`. If the argument is not a `dateTime` or `date` data type, it returns a null value.
IF(condition , expression1, expression2)	Evaluates the condition and obtains the effective Boolean value. If true, the first expression is evaluated and its value returned. If false, the second expression is used. If the condition raises an error, the error is passed as the result of the IF statement.
IRI(RDF term)	Returns an IRI resolving the string representation of argument `term`. If there is a base IRI defined in the query, the IR is resolve against it, and the result must result in an absolute IRI.

Table 1-14 (Cont.) Built-in Functions Available for FILTER Clause

Function	Description
isBLANK(RDF term)	isBLANK(x) returns `true` if `x` is a blank node, `false` otherwise.
isIRI(RDF term)	isIRI(x) returns `true` if `x` is an IRI, `false` otherwise.
isLITERAL(RDF term)	isLiteral(x) returns `true` if `x` is a literal, `false` otherwise.
IsNUMERIC(RDF term)	Returns `true` if `term` is a numeric value, `false` otherwise.
isURI(RDF term)	isURI(x) returns `true` if `x` is a URI, `false` otherwise.
LANG(literal)	LANG(x) returns a plain literal serializing the language tag of `x`.
LANGMATCHES(literal, literal)	LANGMATCHES(x, y) returns `true` if language tag `x` matches language range `y`, `false` otherwise.
LCASE(literal)	Returns a string where each character in literal is converted to its lowercase correspondent.
MD5(literal)	Returns the checksum for `literal`, corresponding to the MD5 hash function. **Note:** Starting from Oracle Database 21c Release, the use of `MD5` algorithm is deprecated. As this function will be desupported in a future release, it is recommended to replace `MD5` with one of the `SHA` hash functions.
MINUTES(argument)	Returns an integer corresponding to the minutes part of `argument`. If the argument is not a `dateTime` or `date` data type, it returns a null value.
MONTH(argument)	Returns an integer corresponding to the month part of `argument`. If the argument is not a `dateTime` or `date` data type, it returns a null value.
NOT_EXISTS(pattern)	Returns `true` if the pattern does not match the query data set, using the current bindings in the containing group graph pattern and the current active graph. It returns `false` otherwise.
NOW()	Returns an `xsd:dateTime` value corresponding to the current time at the moment of the query execution.
RAND()	Generates a numeric value in the range of [0,1).

Table 1-14 (Cont.) Built-in Functions Available for FILTER Clause

Function	Description
REGEX(string, pattern)	REGEX(x,y) returns `true` if `x` matches the regular expression `y`, `false` otherwise. For more information about the regular expressions supported, see the Oracle Regular Expression Support appendix in *Oracle Database SQL Language Reference*.
REGEX(string, pattern, flags)	REGEX(x,y,z) returns `true` if `x` matches the regular expression `y` using the options given in `z`, `false` otherwise. Available options: `'s'` – dot all mode (`'.'` matches any character including the newline character); `'m'` – multiline mode (`'^'` matches the beginning of any line and `'$'` matches the end of any line); `'i'` – case insensitive mode; `'x'` – remove whitespace characters from the regular expression before matching.
REPLACE(string, pattern, replacement)	Returns a string where each match of the regular expression `pattern` in `string` is replaced by `replacement`. For more information about the regular expressions supported, see the Oracle Regular Expression Support appendix in *Oracle Database SQL Language Reference*.
REPLACE(string, pattern, replacement, flags)	Returns a string where each match of the regular expression `pattern` in string is replaced by `replacement`. Available options: `'s'` – dot all mode (`'.'` matches any character including the newline character); `'m'` – multiline mode (`'^'` matches the beginning of any line and `'$'` matches the end of any line); `'i'` – case insensitive mode; `'x'` – remove whitespace characters from the regular expression before matching. For more information about the regular expressions supported, see the Oracle Regular Expression Support appendix in *Oracle Database SQL Language Reference*.
ROUND(RDF term)	Returns the closest number with no fractional part to `term`. If two values exist, the value closer to positive infinite is returned. If term is a non-numerical value, returns null.
sameTerm(RDF term, RDF term)	sameTerm(x, y) returns `true` if `x` and `y` are the same RDF term, `false` otherwise.
SECONDS(argument)	Returns an integer corresponding to the seconds part of `argument`. If the argument is not a `dateTime` or `date` data type, it returns a null value.
SHA1(literal)	Returns the checksum for `literal`, corresponding to the SHA1 hash function.
SHA256(literal)	Returns the checksum for `literal`, corresponding to the SHA256 hash function.

Table 1-14 (Cont.) Built-in Functions Available for FILTER Clause

Function	Description
SHA384(literal)	Returns the checksum for `literal`, corresponding to the SHA384 hash function.
SHA512(literal)	Returns the checksum for `literal`, corresponding to the SHA512 hash function.
STR(RDF term)	STR(x) returns a plain literal of the string representation of `x` (that is, what would be stored in the VALUE_NAME column of RDF_VALUE$ enclosed within double quotes).
STRAFTER(literal, literal)	StrAfter (x,y) returns the portion of the string corresponding to substring that precedes in `x` the first match of `y`, and the end of `x`. If `y` cannot be matched inside `x`, the empty string is returned.
STRBEFORE(literal, literal)	StrBefore (x,y) returns the portion of the string corresponding to the start of `x` and the first match of `y`. If `y` cannot be matched inside `x`, the empty string is returned.
STRDT(string, datatype)	Construct a literal term composed by the `string` lexical form and the `datatype` passed as arguments. `datatype` must be a URI; otherwise, the function returns a null value.
STRENDS(literal, match)	Returns `true` if the string `literal` ends with the string `match`. It returns `false` otherwise.
STRLANG (string, languageTag)	Constructs a string composed by the `string` lexical form and language tag passed as arguments.
STRLEN(literal)	Returns the length of the lexical form of the `literal`.
STRSTARTS(literal, match)	Returns `true` if the string `literal` starts with the string `match`. It returns `false` otherwise.
STRUUID()	Returns a string containing the scheme section of a new UUID.
SUBSTR(term, startPos)	Returns the string corresponding to the portion of `term` that starts at `startPos` and continues until term ends. The index of the first character is 1.
SUBSTR(term, startPos, length)	Returns the string corresponding to the portion of term that starts at `startPos` and continues for `length` characters. The index of the first character is 1.
term IN (term list)	The expression x IN(term list) returns `true` if `x` can be found in any of the values in `termlist`. Returns `false` if not found. Zero-length lists are legal. An error is raised if any of the values in `termlist` raises an error and `x` is not found.
term NOT IN (term list)	The expression x NOT IN(term list) returns `false` if `x` can be found in any of the values in term list. Returns `true` if not found. Zero-length lists are legal. An error is raised if any of the values in term list raises an error and `x` is not found.

Table 1-14 (Cont.) Built-in Functions Available for FILTER Clause

Function	Description
TIMEZONE(argument)	Returns the time zones section of `argument` as an `xsd:dayTimeDuration` value. If the argument is not a `dateTime` or `date` data type, it returns a null value.
TZ(argument)	Returns an integer corresponding to the time zone part of `argument`. If the argument is not a `dateTime` or `date` data type, it returns a null value.
UCASE(literal)	Returns a string where each character in `literal` is converted to its uppercase correspondent.
URI(RDF term)	(Synonym for IRI(RDF term))
UUID()	Returns a URI with a new Universal Unique Identifier. The value and the version correspond to the PL/SQL function `sys_guid ()`.
YEAR(argument)	Returns an integer corresponding to the year part of `argument`.

See also the descriptions of the built-in functions defined in the SPARQL query language specification (`http://www.w3.org/TR/sparql11-query/`), to better understand the built-in functions available in SEM_MATCH.

In addition, Oracle provides some proprietary query functions that take advantage of Oracle Database features and help improve query performance. The following table lists these Oracle-specific query functions. Note that the built-in namespace prefix `orardf` expands to `<http://xmlns.oracle.com/rdf/>`.

Table 1-15 Oracle-Specific Query Functions

Function	Description
`orardf:concat(RDF term, RDF term, …)`	Returns `true` if the given term matches with the given `like` pattern. Otherwise, the function returns `false`.
`orardf:contains(RDF term, RDF term)`	Returns `true` if the string representation of the first term contains the string representation of the second term as a substring. Otherwise, the function returns `false`.
`orardf:instr(RDF term, RDF term)`	Searches the string representation of the first term for the string representation of the second term. Returns an integer indicating the position of the first character of the occurrence in the first term (the first character in a string is position 1). If the search is unsuccessful, then the returned value is 0.
`orardf:instr(RDF term, RDF term, position)`	Searches the string representation of the first term for the string representation of the second term. Returns an integer indicating the position of the first character of the occurrence in first term (the first character is position 1). If the search is unsuccessful, then the returned value is 0. *Position* is a nonzero interger indicating the character of the string representation of the first term at which to begin the search (the first character is position 1). If *position* is negative, then `orardf:instr` counts and searches backwards from the end of the first term.

Table 1-15 (Cont.) Oracle-Specific Query Functions

Function	Description
`orardf:instr(RDF term, RDF term, position, occurrence)`	Searches the string representation of the first term for the string representation of the second term. Returns an integer indicating the position of the first character of the occurrence in first term (the first character is position 1). If the search is unsuccessful, then the returned value is 0. *Position* is a nonzero interger indicating the character of the string representation of the first term at which to begin the search (the first character is position 1). If *position* is negative, then orardf:instr counts and searches backwards from the end of the first term. *Occurrence* is a positive integer indicating which occurrence of the first term `orardf:instr` should search for.
`orardf:lcase(RDF term)`	Returns a string literal whose lexical form is the lower case of the string representation of the input term.
`orardf:like(RDF term, pattern)`	Returns `true` if the given term matches with the given `like` pattern. Otherwise, the function returns `false`. See Full-Text Search for more information.
`orardf:like(RDF term, pattern, flags)`	Returns `true` if the given term matches with the given `like` pattern using the specified flags. Otherwise, the function returns `false`. Available flags: '`i`' – case insensitive mode. See Full-Text Search for more information.
`orardf:ltrim(RDF term)`	Returns the string representation of the input term with all blank characters removed from the left end.
`orardf:ltrim(RDF term, set)`	Returns the string representation of the input term with all of the characters contained in *set* removed from the left end.
`orardf:rtrim(RDF term)`	Returns the string representation of the input term with all blank characters removed from the right end.
`orardf:rtrim(RDF term, set)`	Returns the string representation of the input term with all of the characters contained in *set* removed from the right end.
`orardf:sameCanonTerm(RDF term, RDF term)`	Returns `true` if two terms represent the same canonical RDF term. Otherwise, the function returns `false`. Allows a VALUE_ID-based comparison, which is more efficient than `sameTerm(?x, ?y) or (?x = ?y).`
`orardf:strafter(RDF term, RDF term)`	Returns the part of the string representation of the first term that follows the first occurrence of the string representation of the second term. If there is no such occurrence, then an empty string literal is returned.
`orardf:strbefore(RDF term, RDF term)`	Returns the part of the string representation of the first term that precedes the first occurrence of the string representation of the second term. If there is no such occurrence, then an empty string literal is returned.
`orardf:strends(RDF term, RDF term)`	Returns `true` if the string representation of the first term contains the string representation of the second term as a trailing substring. Otherwise, the function returns `false`.
`orardf:strlen(RDF term)`	Returns the number of characters in the string representation of the input term.
`orardf:strstarts(RDF term, RDF term)`	Returns `true` if the string representation of the first term contains the string representation of the second term as a leading substring. Otherwise, the function returns `false`.
`orardf:substr(RDF term, start)`	Returns a portion of the string representation of the input term beginning at the position indicated by the *start* value and continuing to the end of the input term (the first character is position 1).

Table 1-15 (Cont.) Oracle-Specific Query Functions

Function	Description
`orardf:substr(RDF term, start, length)`	Returns a portion of the string representation of the input term beginning at the position indicated by the *start* value and continuing for the number of characters indicated by the *length* value (the first character is position 1).
`orardf:textContains(RDF term, pattern)`	Returns `true` if the given term matches with the given Oracle Text search pattern. Otherwise, the function returns `false`. See Full-Text Search for more information.
`orardf:textScore(invocation id)`	Returns the score of an `orardf:textContains` match. See Full-Text Search for more information.
`orardf:ucase(RDF term)`	Returns a string literal whose lexical form is the upper case of the lexical form of the input term.
(Spatial built-in functions)	(See Spatial Support.)

The following XML Schema casting functions are available for use in FILTER clauses. These functions take an RDF term as input and return a new RDF term of the desired type or raise an error if the term cannot be cast to the desired type. Details of type casting can be found in Section 17.1 of the XPath query specification: `http://www.w3.org/TR/xpath-functions/#casting-from-primitive-to-primitive`. These functions use the XML Namespace `xsd : http://www.w3.org/2001/XMLSchema#`.

- xsd:string (RDF term)
- xsd:dateTime (RDF term)
- xsd:boolean (RDF term)
- xsd:integer (RDF term)
- xsd:float (RDF term)
- xsd:double (RDF term)
- xsd:decimal (RDF term)

If you use the syntax with curly braces to express a graph pattern:

- The query always returns canonical lexical forms for the matching values for the variables.
- Any hints specified in the `options` argument using HINT0={<*hint-string*>} (explained in Using the SEM_MATCH Table Function to Query RDF Data), should be constructed only on the basis of the portion of the graph pattern inside the root BGP. For example, the only valid aliases for use in a hint specification for the query in Example 1-16 are `t0`, `t1`, `?x`, and `?y`. Inline query optimizer hints can be used to influence other parts of the graph pattern (see Inline Query Optimizer Hints).
- The FILTER construct is not supported for variables bound to long literals.

Example 1-15 Curly Brace Syntax

Example 1-15 uses the syntax with curly braces and a period to express a graph pattern in the SEM_MATCH table function.

```
SELECT x, y
  FROM TABLE(SEM_MATCH(
    '{?x :grandParentOf ?y . ?x rdf:type :Male}',
    SEM_Models('family'),
    SEM_Rulebases('RDFS','family_rb'),
```

```
    SEM_ALIASES(SEM_ALIAS('','http://www.example.org/family/')),
    null, null, '', null, null,
    'RDFUSER', 'NET1'));
```

Example 1-16 Curly Brace Syntax and OPTIONAL Construct

Example 1-16 uses the OPTIONAL construct to modify Example 1-15, so that it also returns, for each grandfather, the names of the games played or null if no games are played.

```
SELECT x, y, game
  FROM TABLE(SEM_MATCH(
    '{?x :grandParentOf ?y . ?x rdf:type :Male .
      OPTIONAL{?x :plays ?game}
     }',
    SEM_Models('family'),
    SEM_Rulebases('RDFS','family_rb'),
    SEM_ALIASES(SEM_ALIAS('','http://www.example.org/family/')),
    null,
    null,
    'HINT0={LEADING(t0 t1) USE_NL(?x ?y)}',
    null,
    null,
    'RDFUSER', 'NET1'));
```

Example 1-17 Curly Brace Syntax and Multi-Pattern OPTIONAL Construct

When multiple triple patterns are present in an OPTIONAL graph pattern, values for optional variables are returned only if a match is found for each triple pattern in the OPTIONAL graph pattern. Example 1-17 modifies Example 1-16 so that it returns, for each grandfather, the names of the games played with the grandchildren, or null if they have no such games in common. It also uses global query optimizer hints to specify that triple patterns should be evaluated in order within each BGP and that a hash join should be used to join the root BGP with the OPTIONAL BGP.

```
SELECT x, y, game
  FROM TABLE(SEM_MATCH(
    '{?x :grandParentOf ?y . ?x rdf:type :Male .
      OPTIONAL{?x :plays ?game . ?y :plays ?game}
     }',
    SEM_Models('family'),
    SEM_Rulebases('RDFS','family_rb'),
    SEM_ALIASES(SEM_ALIAS('','http://www.example.org/family/')),
    null, null,
    'ALL_ORDERED ALL_BGP_HASH',
    null, null,
    'RDFUSER', 'NET1'));
```

Example 1-18 Curly Brace Syntax and Nested OPTIONAL Construct

A single query can contain multiple OPTIONAL graph patterns, which can be nested or parallel. Example 1-18 modifies Example 1-17 with a nested OPTIONAL graph pattern. This example returns (1) the games each grandfather plays or null if they play no games and (2) if the grandfather plays games, the ages of the grandchildren that play the same game, or null if they has no games in common. Note that in Example 1-18 a value is returned for `?game` even if the nested OPTIONAL graph pattern `?y :plays ?game . ?y :age ?age` is not matched.

```
SELECT x, y, game, age
  FROM TABLE(SEM_MATCH(
    '{?x :grandParentOf ?y . ?x rdf:type :Male .
      OPTIONAL{?x :plays ?game
                         OPTIONAL {?y :plays ?game . ?y :age ?age} }
     }',
```

```
    SEM_Models('family'),
    SEM_Rulebases('RDFS','family_rb'),
    SEM_ALIASES(SEM_ALIAS('','http://www.example.org/family/')),
    null, null, ' ', null, null,
    'RDFUSER', 'NET1'));
```

Example 1-19   Curly Brace Syntax and Parallel OPTIONAL Construct

Example 1-19 modifies Example 1-17 with a parallel OPTIONAL graph pattern. This example returns (1) the games the each grandfather plays or null if they play no games and (2) the email address of each grandfather or null if they have no email address. Note that, unlike nested OPTIONAL graph patterns, parallel OPTIONAL graph patterns are treated independently. That is, if an email address is found, it will be returned regardless of whether or not a game was found; and if a game was found, it will be returned regardless of whether an email address was found.

```
SELECT x, y, game, email
  FROM TABLE(SEM_MATCH(
    '{?x :grandParentOf ?y . ?x rdf:type :Male .
      OPTIONAL{?x :plays ?game}
      OPTIONAL{?x :email ?email}
     }',
    SEM_Models('family'),
    SEM_Rulebases('RDFS','family_rb'),
    SEM_ALIASES(SEM_ALIAS('','http://www.example.org/family/')),
    null, null, ' ', null, null,
    'RDFUSER', 'NET1'));
```

Example 1-20   Curly Brace Syntax and FILTER Construct

Example 1-20 uses the FILTER construct to modify Example 1-15, so that it returns grandchildren information for only those grandfathers who are residents of either NY or CA.

```
SELECT x, y
  FROM TABLE(SEM_MATCH(
    '{?x :grandParentOf ?y . ?x rdf:type :Male . ?x :residentOf ?z
       FILTER (?z = "NY"  || ?z = "CA")}',
    SEM_Models('family'),
    SEM_Rulebases('RDFS','family_rb'),
    SEM_ALIASES(SEM_ALIAS('','http://www.example.org/family/')),
    null, null, ' ', null, null,
    'RDFUSER', 'NET1'));
```

Example 1-21   Curly Brace Syntax and FILTER with REGEX and STR Built-In Constructs

Example 1-21 uses the REGEX built-in function to select all grandfathers who have an Oracle email address. Note that backslash (\) characters in the regular expression pattern must be escaped in the query string; for example, `\\.` produces the following pattern: `\.`

```
SELECT x, y, z
  FROM TABLE(SEM_MATCH(
    '{?x :grandParentOf ?y . ?x rdf:type :Male . ?x :email ?z
       FILTER (REGEX(STR(?z), "@oracle\\.com$"))}',
    SEM_Models('family'),
    SEM_Rulebases('RDFS','family_rb'),
    SEM_ALIASES(SEM_ALIAS('','http://www.example.org/family/')),
    null, null, ' ', null, null,
    'RDFUSER', 'NET1'));
```

Example 1-22 Curly Brace Syntax and UNION and FILTER Constructs

Example 1-22 uses the UNION construct to modify Example 1-20, so that grandfathers are returned only if they are residents of NY or CA or own property in NY or CA, or if both conditions are true (they reside in and own property in NY or CA).

```
SELECT x, y
  FROM TABLE(SEM_MATCH(
    '{?x :grandParentOf ?y . ?x rdf:type :Male
       {{?x :residentOf ?z} UNION {?x :ownsPropertyIn ?z}}
       FILTER (?z = "NY"  || ?z = "CA")}',
    SEM_Models('family'),
    SEM_Rulebases('RDFS','family_rb'),
    SEM_ALIASES(SEM_ALIAS('','http://www.example.org/family/')),
    null, null, ' ', null, null,
    'RDFUSER', 'NET1'));
```

- GRAPH Keyword Support

1.7.2.1 GRAPH Keyword Support

A SEM_MATCH query is executed against an RDF Dataset. An RDF Dataset is a collection of graphs that includes one unnamed graph, known as the default graph, and one or more named graphs, which are identified by a URI. Graph patterns that appear inside a GRAPH clause are matched against the set of named graphs, and graph patterns that do not appear inside a graph clause are matched against the default graph. The `graphs` and `named_graphs` SEM_MATCH parameters are used to construct the default graph and set of named graphs for a given SEM_MATCH query. A summary of possible dataset configurations is shown in Table 1-16.

Table 1-16 SEM_MATCH graphs and named_graphs Values, and Resulting Dataset Configurations

Parameter Values	Default Graph	Set of Named Graphs
`graphs`: NULL `named_graphs`: NULL	Union All of all unnamed triples and all named graph triples. (But if the `options` parameter contains `STRICT_DEFAULT=T`, only unnamed triples are included in the default graph.)	All named graphs
`graphs`: NULL `named_graphs`: {g1,…, gn}	Empty set	{g1,…, gn}
`graphs`: {g1,…, gm} `named_graphs`: NULL	Union All of {g1,…, gm}	Empty set
`graphs`: {g1,…, gm} `named_graphs`: {gn,…, gz}	Union All of {g1,…, gm}	{gn,…, gz}

See also the W3C SPARQL specification for more information on RDF data sets and the GRAPH construct, specifically: `http://www.w3.org/TR/rdf-sparql-query/#rdfDataset`

Example 1-23 Named Graph Construct

Example 1-23 uses the GRAPH construct to scope graph pattern matching to a specific named graph. This example finds the names and email addresses of all people in the `<http://www.example.org/family/Smith>` named graph.

```
SELECT name, email
  FROM TABLE(SEM_MATCH(
    '{GRAPH :Smith {
       ?x :name ?name . ?x :email ?email } }',
```

```
    SEM_Models('family'),
    SEM_Rulebases('RDFS','family_rb'),
    SEM_ALIASES(SEM_ALIAS('','http://www.example.org/family/')),
    null, null, ' ', null, null,
    'RDFUSER', 'NET1'));
```

Example 1-24 Using the named_graphs Parameter

In addition to URIs, variables can appear after the GRAPH keyword. Example 1-24 uses a variable, `?g`, with the GRAPH keyword, and uses the `named_graphs` parameter to restrict the possible values of `?g` to the `<http://www.example.org/family/Smith>` and `<http://www.example.org/family/Jones>` named graphs. Aliases specified in SEM_ALIASES argument can be used in the `graphs` and `named_graphs` parameters.

```
SELECT name, email
  FROM TABLE(SEM_MATCH(
    '{GRAPH ?g {
       ?x :name ?name . ?x :email ?email } }',
    SEM_Models('family'),
    SEM_Rulebases('RDFS','family_rb'),
    SEM_ALIASES(SEM_ALIAS('','http://www.example.org/family/')),
    null,null,null,null,
    SEM_GRAPHS('<http://www.example.org/family/Smith>',
               ':Jones'),
    'RDFUSER', 'NET1'));
```

Example 1-25 Using the graphs Parameter

Example 1-25 uses the default graph to query the union of the `<http://www.example.org/family/Smith>` and `<http://www.example.org/family/Jones>` named graphs.

```
FROM TABLE(SEM_MATCH(
    '{?x :name ?name . ?x :email ?email }',
    SEM_Models('family'),
    SEM_Rulebases('RDFS','family_rb'),
    SEM_ALIASES(SEM_ALIAS('','http://www.example.org/family/')),
    null,null,null,
    SEM_GRAPHS('<http://www.example.org/family/Smith>',
               ':Jones'),
    null,
    'RDFUSER', 'NET1'));
```

1.7.3 Graph Patterns: Support for SPARQL ASK Syntax

SEM_MATCH allows fully-specified SPARQL ASK queries in the query parameter.

ASK queries are used to test whether or not a solution exists for a given query pattern. In contrast to other forms of SPARQL queries, ASK queries do not return any information about solutions to the query pattern. Instead, such queries return `"true"^^xsd:boolean` if a solution exists and `"false"^^xsd:boolean` if no solution exists.

All SPARQL ASK queries return the same columns: ASK, ASK$RDFVID, ASK$_PREFIX, ASK$_SUFFIX, ASK$RDFVTYP, ASK$RDFCLOB, ASK$RDFLTYP, ASK$RDFLANG, SEM$ROWNUM. Note that these columns are the same as a SPARQL SELECT syntax query that projects a single `?ask` variable.

SPARQL ASK queries will generally give better performance than an equivalent SPARQL SELECT syntax query because the ASK query does not have to retrieve lexical values for query variables, and query execution can stop after a single result has been found.

SPARQL ASK queries use the same syntax as SPARQL SELECT queries, but the topmost SELECT clause must be replaced with the keyword ASK.

Example 1-26   SPARQL ASK

Example 1-26 shows a SPARQL ASK query that determines whether or not any cameras are for sale with more than 10 megapixels that cost less than 50 dollars.

```
SELECT ask
  FROM TABLE(SEM_MATCH(
    'PREFIX : <http://www.example.org/electronics/>
     ASK
     WHERE
      {?x :price ?p .
       ?x :megapixels ?m .
       FILTER (?p < 50 && ?m > 10)
      }',
    SEM_Models('electronics'),
    null, null, null, ' ', null, null,
    'RDFUSER', 'NET1'));
```

See also the W3C SPARQL specification for more information on SPARQL ASK queries, specifically: `http://www.w3.org/TR/sparql11-query/#ask`

1.7.4 Graph Patterns: Support for SPARQL CONSTRUCT Syntax

SEM_MATCH allows fully-specified SPARQL CONSTRUCT queries in the query parameter.

CONSTRUCT queries are used to build RDF graphs from stored RDF data. In contrast to SPARQL SELECT queries, CONSTRUCT queries return a set of RDF triples rather than a set of query solutions (variable bindings).

All SPARQL CONSTRUCT queries return the same columns from SEM_MATCH. These columns correspond to the subject, predicate and object of an RDF triple, and there are 10 columns for each triple component. In addition, a SEM$ROWNUM column is returned. More specifically, the following columns are returned:

```
SUBJ
SUBJ$RDFVID
SUBJ$_PREFIX
SUBJ$_SUFFIX
SUBJ$RDFVTYP
SUBJ$RDFCLOB
SUBJ$RDFLTYP
SUBJ$RDFLANG
SUBJ$RDFTERM
SUBJ$RDFCLBT
PRED
PRED$RDFVID
PRED$_PREFIX
PRED$_SUFFIX
PRED$RDFVTYP
PRED$RDFCLOB
PRED$RDFLTYP
PRED$RDFLANG
PRED$RDFTERM
PRED$RDFCLBT
OBJ
OBJ$RDFVID
OBJ$_PREFIX
OBJ$_SUFFIX
```

```
OBJ$RDFVTYP
OBJ$RDFCLOB
OBJ$RDFLTYP
OBJ$RDFLANG
OBJ$RDFTERM
OBJ$RDFCLBT
SEM$ROWNUM
```

For each component, COMP, COMP$RDFVID, COMP$_PREFIX, COMP$_SUFFIX, COMP$RDFVTYP, COMP$RDFCLOB, COMP$RDFLTYP, and COMP$RDFLANG correspond to the same values as those from SPARQL SELECT queries. COMP$RDFTERM holds a VARCHAR2(`NETWORK_MAX_STRING_SIZE`) RDF term in N-Triple syntax, and COMP$RDFCLBT holds a CLOB RDF term in N-Triple syntax.

SPARQL CONSTRUCT queries use the same syntax as SPARQL SELECT queries, except the topmost SELECT clause is replaced with a CONSTRUCT template. The CONSTRUCT template determines how to construct the result RDF graph using the results of the query pattern defined in the WHERE clause. A CONSTRUCT template consists of the keyword CONSTRUCT followed by sequence of SPARQL triple patterns that are enclosed within curly braces. The keywords OPTIONAL, UNION, FILTER, MINUS, BIND, VALUES, and GRAPH are not allowed within CONSTRUCT templates, and property path expressions are not allowed within CONSTRUCT templates. These keywords, however, are allowed within the query pattern inside the WHERE clause.

SPARQL CONSTRUCT queries build result RDF graphs in the following manner. For each result row returned by the WHERE clause, variable values are substituted into the CONSTRUCT template to create one or more RDF triples. Suppose the graph pattern in the WHERE clause of Example 1-27 returns the following result rows.

E$RDFTERM	FNAME$RDFTERM	LNAME$RDFTERM
ent:employee1	"Fred"	"Smith"
ent:employee2	"Jane"	"Brown"
ent:employee3	"Bill"	"Jones"

The overall SEM_MATCH CONSTRUCT query in Example 1-27 would then return the following rows, which correspond to six RDF triples (two for each result row of the query pattern).

SUBJ$RDFTERM	PRED$RDFTERM	OBJ$RDFTERM
ent:employee1	foaf:givenName	"Fred"
ent:employee1	foaf:familyName	"Smith"
ent:employee2	foaf:givenName	"Jane"
ent:employee2	foaf:familyName	"Brown"
ent:employee3	foaf:givenName	"Bill"
ent:employee3	foaf:familyName	"Jones"

There are two SEM_MATCH query options that influence the behavior of SPARQL CONSTRUCT: `CONSTRUCT_UNIQUE=T` and `CONSTRUCT_STRICT=T`. Using the `CONSTRUCT_UNIQUE=T` query option ensures that only unique RDF triples are returned from the CONSTRUCT query. Using the `CONSTRUCT_STRICT=T` query option ensures that only valid RDF triples are returned from the CONSTRUCT query. Valid RDF triples are those that have (1) a URI or blank node in the subject position, (2) a URI in the predicate position, and (3) a URI, blank node or RDF literal in the object position. Both of these query options are turned off by default for improved query performance.

Example 1-27 SPARQL CONSTRUCT

Example 1-27 shows a SPARQL CONSTRUCT query that builds an RDF graph of employee names using the foaf vocabulary.

```
SELECT subj$rdfterm, pred$rdfterm, obj$rdfterm
  FROM TABLE(SEM_MATCH(
    'PREFIX  ent: <http://www.example.org/enterprise/>
     PREFIX foaf: <http://xmlns.com/foaf/0.1/>
     CONSTRUCT
      {?e foaf:givenName  ?fname .
       ?e foaf:familyName ?lname
      }
     WHERE
      {?e ent:fname ?fname .
       ?e ent:lname ?lname
      }',
    SEM_Models('enterprise'),
    SEM_Rulebases('RDFS'),
    null, null, null, ' ', null, null,
    'RDFUSER', 'NET1'));
```

Example 1-28 CONSTRUCT with Solution Modifiers

SPARQL SOLUTION modifiers can be used with CONSTRUCT queries. Example 1-28 shows the use of ORDER BY and LIMIT to build a graph about the top two highest-paid employees. Note that the LIMIT 2 clause applies to the query pattern not to the overall CONSTRUCT query. That is, the query pattern will return two result rows, but the overall CONSTRUCT query will return 6 RDF triples (three for each of the two employees bound to `?e`).

```
SELECT subj$rdfterm, pred$rdfterm, obj$rdfterm
  FROM TABLE(SEM_MATCH(
    'PREFIX  ent: <http://www.example.org/enterprise/>
     PREFIX foaf: <http://xmlns.com/foaf/0.1/>
     CONSTRUCT
      { ?e ent:fname       ?fname .
        ?e ent:lname       ?lname .
        ?e ent:dateOfBirth ?dob }
     WHERE
      { ?e ent:fname  ?fname .
        ?e ent:lname  ?lname .
        ?e ent:salary ?sal
      }
     ORDER BY DESC(?sal)
     LIMIT 2',
    SEM_Models('enterprise'),
    SEM_Rulebases('RDFS'),
    null, null, null, ' ', null, null,
    'RDFUSER', 'NET1'));
```

Example 1-29 SPARQL 1.1 Features with CONSTRUCT

SPARQL 1.1 features are supported within CONSTRUCT query patterns. Example 1-29 shows the use of subqueries and SELECT expressions within a CONSTRUCT query.

```
SELECT subj$rdfterm, pred$rdfterm, obj$rdfterm
  FROM TABLE(SEM_MATCH(
    'PREFIX  ent: <http://www.example.org/enterprise/>
     PREFIX foaf: <http://xmlns.com/foaf/0.1/>
     CONSTRUCT
```

```
      { ?e foaf:name  ?name }
     WHERE
      { SELECT ?e (CONCAT(?fname," ",?lname) AS ?name)
        WHERE { ?e ent:fname ?fname .
                ?e ent:lname ?lname }
      }',
    SEM_Models('enterprise'),
    SEM_Rulebases('RDFS'),
    null, null, null, ' ', null, null,
    'RDFUSER', 'NET1'));
```

Example 1-30 SPARQL CONSTRUCT with Named Graphs

Named graph data cannot be returned from SPARQL CONSTRUCT queries because, in accordance with the W3C SPARQL specification, only RDF triples are returned, not RDF quads. The FROM, FROM NAMED and GRAPH keywords, however, can be used when matching the query pattern defined in the WHERE clause.

Example 1-30 constructs an RDF graph with `ent:name` triples from the UNION of named graphs `ent:g1` and `ent:g2`, `ent:dateOfBirth` triples from named graph `ent:g3`, and `ent:ssn` triples from named graph `ent:g4`.

```
SELECT subj$rdfterm, pred$rdfterm, obj$rdfterm
  FROM TABLE(SEM_MATCH(
    'PREFIX  ent: <http://www.example.org/enterprise/>
     PREFIX foaf: <http://xmlns.com/foaf/0.1/>
     CONSTRUCT
      { ?e ent:name ?name .
        ?e ent:dateOfBirth ?dob .
        ?e ent:ssn ?ssn
      }
     FROM ent:g1
     FROM ent:g2
     FROM NAMED ent:g3
     FROM NAMED ent:g4
     WHERE
      { ?e foaf:name ?name .
        GRAPH ent:g3 { ?e ent:dateOfBirth ?dob }
        GRAPH ent:g4 { ?e ent:ssn ?ssn }
      }',
    SEM_Models('enterprise'),
    SEM_Rulebases('RDFS'),
    null, null, null, ' ', null, null,
    'RDFUSER', 'NET1'));
```

Example 1-31 SPARQL CONSTRUCT Normal Form

```
SELECT subj$rdfterm, pred$rdfterm, obj$rdfterm
  FROM TABLE(SEM_MATCH(
    'PREFIX  ent: <http://www.example.org/enterprise/>
     PREFIX foaf: <http://xmlns.com/foaf/0.1/>
     CONSTRUCT
      {?e foaf:givenName  ?fname .
       ?e foaf:familyName ?lname
      }
     WHERE
      {?e ent:fname ?fname .
       ?e ent:lname ?lname
      }',
    SEM_Models('enterprise'),
    SEM_Rulebases('RDFS'),
```

```
    null, null, null, ' ', null, null,
    'RDFUSER', 'NET1'));
```

Example 1-32 SPARQL CONSTRUCT Short Form

A short form of CONSTRUCT is supported when the CONSTRUCT template is exactly the same as the WHERE clause. In this case, only the keyword CONSTRUCT is needed, and the graph pattern in the WHERE clause will also be used as a CONSTRUCT template. Example 1-32 shows the short form of Example 1-31.

```
SELECT subj$rdfterm, pred$rdfterm, obj$rdfterm
  FROM TABLE(SEM_MATCH(
    'PREFIX  ent: <http://www.example.org/enterprise/>
     PREFIX foaf: <http://xmlns.com/foaf/0.1/>
     CONSTRUCT
     WHERE
      {?e ent:fname ?fname .
       ?e ent:lname ?lname
      }',
    SEM_Models('enterprise'),
    SEM_Rulebases('RDFS'),
    null, null, null, ' ', null, null,
    'RDFUSER', 'NET1'));
```

- Typical SPARQL CONSTRUCT Workflow

1.7.4.1 Typical SPARQL CONSTRUCT Workflow

A typical workflow for SPARQL CONSTRUCT would be to execute a CONSTRUCT query to extract and/or transform RDF triple data from an existing RDF graph and then load this data into an existing or new RDF graph. The data loading can be accomplished through simple INSERT statements or executing the SEM_APIS.BULK_LOAD_RDF_GRAPH procedure.

Example 1-33 SPARQL CONSTRUCT Workflow

Example 1-33 constructs `foaf:name` triples from existing `ent:fname` and `ent:lname` triples and then bulk loads these new triples back into the original RDF graph. Afterward, you can query the original graph for `foaf:name` values.

```
-- Use create table as select to build a staging table
CREATE TABLE STAB(RDF$STC_sub, RDF$STC_pred, RDF$STC_obj) AS
SELECT subj$rdfterm, pred$rdfterm, obj$rdfterm
FROM TABLE(SEM_MATCH(
 'PREFIX  ent: <http://www.example.org/enterprise/>
  PREFIX foaf: <http://xmlns.com/foaf/0.1/>
  CONSTRUCT
   { ?e foaf:name  ?name }
  WHERE
   { SELECT ?e (CONCAT(?fname," ",?lname) AS ?name)
     WHERE { ?e ent:fname ?fname .
             ?e ent:lname ?lname }
   }',
 SEM_Models('enterprise'),
 null, null, null, null, ' ', null, null,
 'RDFUSER', 'NET1'));



-- Bulk load data back into the enterprise model
BEGIN
  SEM_APIS.BULK_LOAD_RDF_GRAPH(
```

```
    rdf_graph_name=>'enterprise',
    table_owner=>'rdfuser',
    table_name=>'stab',
    flags=>' parallel_create_index parallel=4 ',
    network_owner=>'RDFUSER',
    network_name=>'NET1');
END;
/

-- Query for foaf:name data
SELECT e$rdfterm, name$rdfterm
FROM TABLE(SEM_MATCH(
 'PREFIX foaf: <http://xmlns.com/foaf/0.1/>
  SELECT ?e ?name
  WHERE { ?e foaf:name ?name }',
 SEM_Models('enterprise'),
 null, null, null, null, ' ', null, null,
 'RDFUSER', 'NET1'));
```

See also the W3C SPARQL specification for more information on SPARQL CONSTRUCT queries, specifically: `http://www.w3.org/TR/sparql11-query/#construct`

1.7.5 Graph Patterns: Support for SPARQL DESCRIBE Syntax

SEM_MATCH allows fully-specified SPARQL DESCRIBE queries in the query parameter.

SPARQL DESCRIBE queries are useful for exploring RDF data sets. You can easily find information about a given resource or set of resources without knowing information about the exact RDF properties used in the data set. A DESCRIBE query returns a "description" of a resource `r`, where a "description" is the set of RDF triples in the query data set that contain `r` in either the subject or object position.

Like CONSTRUCT queries, DESCRIBE queries return an RDF graph instead of result bindings. Each DESCRIBE query, therefore, returns the same columns as a CONSTRUCT query (see Graph Patterns: Support for SPARQL CONSTRUCT Syntax for a listing of return columns).

SPARQL DESCRIBE queries use the same syntax as SPARQL SELECT queries, except the topmost SELECT clause is replaced with a DESCRIBE clause. A DESCRIBE clause consists of the DESCRIBE keyword followed by a sequence of URIs and/or variables separated by whitespace or the DESCRIBE keyword followed by a single * (asterisk).

Two SEM_MATCH query options affect SPARQL DESCRIBE queries: `CONSTRUCT_UNIQUE=T` and `CONSTRUCT_STRICT=T`. `CONSTRUCT_UNIQUE=T` ensures that duplicate triples are eliminated from the result, and `CONSTRUCT_STRICT=T` ensures that invalid triples are eliminated from the result. Both of these options are turned off by default. These options are described in more detail in Graph Patterns: Support for SPARQL CONSTRUCT Syntax.

See also the W3C SPARQL specification for more information on SPARQL DESCRIBE queries, specifically: `http://www.w3.org/TR/sparql11-query/#describe`

Example 1-34 SPARQL DESCRIBE Short Form

A short form of SPARQL DESCRIBE is provided to describe a single constant URI. In the short form, only a DESCRIBE clause is needed. Example 1-34 shows a short form SPARQL DESCRIBE query.

```
SELECT subj$rdfterm, pred$rdfterm, obj$rdfterm
FROM TABLE(SEM_MATCH(
 'DESCRIBE <http://www.example.org/enterprise/emp_1>',
```

```
 SEM_Models('enterprise'),
 null, null, null, null, ' ', null, null,
 'RDFUSER', 'NET1'));
```

Example 1-35 SPARQL DESCRIBE Normal Form

The normal form of SPARQL DESCRIBE specifies a DESCRIBE clause and a SPARQL query pattern, possibly including solution modifiers. Example 1-35 shows a SPARQL DESCRIBE query that describes all employees whose departments are located in New Hampshire.

```
SELECT subj$rdfterm, pred$rdfterm, obj$rdfterm
FROM TABLE(SEM_MATCH(
 'PREFIX  ent: <http://www.example.org/enterprise/>
  DESCRIBE ?e
  WHERE
   { ?e ent:department ?dept .
     ?dept ent:locatedIn "New Hampshire" }',
 SEM_Models('enterprise'),
 null, null, null, null, ' ', null, null,
 'RDFUSER', 'NET1'));
```

Example 1-36 DESCRIBE *

With the normal form of DESCRIBE, as shown in Example 1-35, all resources bound to variables listed in the DESCRIBE clause are described. In Example 1-35, all employees returned from the query pattern and bound to `?e` will be described. When DESCRIBE * is used, all visible variables in the query are described.

Example 1-36 shows a modified version of Example 1-35 that describes both employees (bound to `?e`) and departments (bound to `?dept`).

```
SELECT subj$rdfterm, pred$rdfterm, obj$rdfterm
FROM TABLE(SEM_MATCH(
 'PREFIX  ent: <http://www.example.org/enterprise/>
  DESCRIBE *
  WHERE
   { ?e ent:department ?dept .
     ?dept ent:locatedIn "New Hampshire" }',
 SEM_Models('enterprise'),
 null, null, null, null, ' ', null, null,
 'RDFUSER', 'NET1'));
```

1.7.6 Graph Patterns: Support for SPARQL SELECT Syntax

In addition to curly-brace graph patterns, SEM_MATCH allows fully-specified SPARQL SELECT queries in the `query` parameter. When using the SPARQL SELECT syntax option, SEM_MATCH supports the following query constructs: BASE, PREFIX, SELECT, SELECT DISTINCT, FROM, FROM NAMED, WHERE, ORDER BY, LIMIT, and OFFSET. Each SPARQL SELECT syntax query must include a SELECT clause and a graph pattern.

A key difference between curly-brace and SPARQL SELECT syntax when using SEM_MATCH is that only variables appearing in the SPARQL SELECT clause are returned from SEM_MATCH when using SPARQL SELECT syntax.

One additional column, SEM$ROWNUM, is returned from SEM_MATCH when using SPARQL SELECT syntax. This NUMBER column can be used to order the results of a SEM_MATCH query so that the result order matches the ordering specified by a SPARQL ORDER BY clause.

The SPARQL ORDER BY clause can be used to order the results of SEM_MATCH queries. This clause specifies a sequence of comparators used to order the results of a given query. A comparator consists of an expression composed of variables, RDF terms, arithmetic operators

(+, -, *, /), Boolean operators and logical connectives (||, &&, !), comparison operators (<, >, <=, >=, =, !=), and any functions available for use in FILTER expressions.

The following order of operations is used when evaluating SPARQL SELECT queries:

1. Graph pattern matching
2. Grouping (see Grouping and Aggregation.)
3. Aggregates (see Grouping and Aggregation)
4. Having (see Grouping and Aggregation)
5. Values (see Value Assignment)
6. Select expressions
7. Order by
8. Projection
9. Distinct
10. Offset
11. Limit

See also the W3C SPARQL specification for more information on SPARQL BASE, PREFIX, SELECT, SELECT DISTINCT, FROM, FROM NAMED, WHERE, ORDER BY, LIMIT, and OFFSET constructs, specifically: `http://www.w3.org/TR/sparql11-query/`

Example 1-37 SPARQL PREFIX, SELECT, and WHERE Clauses

Example 1-37 uses the following SPARQL constructs:

- SPARQL PREFIX clause to specify an abbreviation for the `http://www.example.org/family/` and `http://xmlns.com/foaf/0.1/` namespaces
- SPARQL SELECT clause to specify the set of variables to project out of the query
- SPARQL WHERE clause to specify the query graph pattern

```
SELECT y, name
  FROM TABLE(SEM_MATCH(
    'PREFIX : <http://www.example.org/family/>
     PREFIX foaf: <http://xmlns.com/foaf/0.1/>
     SELECT ?y ?name
     WHERE
     {?x :grandParentOf ?y .
      ?x foaf:name ?name }',
    SEM_Models('family'),
    SEM_Rulebases('RDFS','family_rb'),
    null, null, null, ' ', null, null,
    'RDFUSER', 'NET1'));
```

Example 1-37 returns the following columns: y, y$RDFVID, y$_PREFIX, y$_SUFFIX, y$RDFVTYP, y$RDFCLOB, y$RDFLTYP, y$RDFLANG, name, name$RDFVID, name$_PREFIX, name$_SUFFIX, name$RDFVTYP, name$RDFCLOB, name$RDFLTYP, name$RDFLANG, and SEM$ROWNUM.

Example 1-38 SPARQL SELECT * (All Variables in Triple Pattern)

The SPARQL SELECT clause specifies either (A) a sequence of variables and/or expressions (see Expressions in the SELECT Clause), or (B) * (asterisk), which projects all variables that appear in a specified triple pattern. Example 1-38 uses the SPARQL SELECT clause to select all variables that appear in a specified triple pattern.

```
SELECT x, y, name
  FROM TABLE(SEM_MATCH(
    'PREFIX : <http://www.example.org/family/>
     PREFIX foaf: <http://xmlns.com/foaf/0.1/>
     SELECT *
     WHERE
     {?x :grandParentOf ?y .
      ?x foaf:name ?name }',
    SEM_Models('family'),
    SEM_Rulebases('RDFS','family_rb'),
    null, null, null, ' ', null, null,
    'RDFUSER', 'NET1'));
```

Example 1-39 SPARQL SELECT DISTINCT

The DISTINCT keyword can be used after SELECT to remove duplicate result rows. Example 1-39 uses SELECT DISTINCT to select only the distinct names.

```
SELECT name
  FROM TABLE(SEM_MATCH(
    'PREFIX : <http://www.example.org/family/>
     PREFIX foaf: <http://xmlns.com/foaf/0.1/>
     SELECT DISTINCT ?name
     WHERE
     {?x :grandParentOf ?y .
      ?x foaf:name ?name }',
    SEM_Models('family'),
    SEM_Rulebases('RDFS','family_rb'),
    null, null, null, ' ', null, null,
    'RDFUSER', 'NET1'));
```

Example 1-40 RDF Dataset Specification Using FROM and FROM NAMED

SPARQL FROM and FROM NAMED are used to specify the RDF dataset for a query. FROM clauses are used to specify the set of graphs that make up the default graph, and FROM NAMED clauses are used to specify the set of graphs that make up the set of named graphs. Example 1-40 uses FROM and FROM NAMED to select email addresses and friend of relationships from the union of the `<http://www.friends.com/friends>` and `<http://www.contacts.com/contacts>` graphs and grandparent information from the `<http://www.example.org/family/Smith>` and `<http://www.example.org/family/Jones>` graphs.

```
SELECT x, y, z, email
  FROM TABLE(SEM_MATCH(
    'PREFIX : <http://www.example.org/family/>
     PREFIX foaf: <http://xmlns.com/foaf/0.1/>
     PREFIX friends: <http://www.friends.com/>
     PREFIX contacts: <http://www.contacts.com/>
     SELECT *
     FROM friends:friends
     FROM contacts:contacts
     FROM NAMED :Smith
     FROM NAMED :Jones
     WHERE
     {?x foaf:frendOf ?y .
      ?x :email ?email .
      GRAPH ?g {
        ?x :grandParentOf ?z }
     }',
    SEM_Models('family'),
    SEM_Rulebases('RDFS','family_rb'),
    null, null, null, ' ', null, null,
    'RDFUSER', 'NET1'));
```

Example 1-41 SPARQL ORDER BY

In a SPARQL ORDER BY clause:

- Single variable ordering conditions do not require enclosing parenthesis, but parentheses are required for more complex ordering conditions.
- An optional ASC() or DESC() order modifier can be used to indicate the desired order (ascending or descending, respectively). Ascending is the default order.
- When using SPARQL ORDER BY in SEM_MATCH, the containing SQL query should be ordered by SEM$ROWNUM to ensure that the desired ordering is maintained through any enclosing SQL blocks.

Example 1-41 uses a SPARQL ORDER BY clause to select all cameras, and it specifies ordering by descending type and ascending total price (`price * (1 - discount) * (1 + tax)`).

```
SELECT *
  FROM TABLE(SEM_MATCH(
    'PREFIX : <http://www.example.org/electronics/>
     SELECT *
     WHERE
      {?x :price ?p .
       ?x :discount ?d .
       ?x :tax ?t .
       ?x :cameraType ?cType .
      }
     ORDER BY DESC(?cType) ASC(?p * (1-?d) * (1+?t))',
    SEM_Models('electronics'),
    SEM_Rulebases('RDFS'),
    null, null, null, ' ', null, null,
    'RDFUSER', 'NET1'))
ORDER BY SEM$ROWNUM;
```

Example 1-42 SPARQL LIMIT

SPARQL LIMIT and SPARQL OFFSET can be used to select different subsets of the query solutions. Example 1-42 uses SPARQL LIMIT to select the five cheapest cameras, and Example 1-43 uses SPARQL LIMIT and OFFSET to select the fifth through tenth cheapest cameras.

```
SELECT *
  FROM TABLE(SEM_MATCH(
    'PREFIX : <http://www.example.org/electronics/>
     SELECT ?x ?cType ?p
     WHERE
      {?x :price ?p .
       ?x :cameraType ?cType .
      }
     ORDER BY ASC(?p)
     LIMIT 5',
    SEM_Models('electronics'),
    SEM_Rulebases('RDFS'),
    null, null, null, ' ', null, null,
    'RDFUSER', 'NET1'))
ORDER BY SEM$ROWNUM;
```

Example 1-43 SPARQL OFFSET

```
SELECT *
  FROM TABLE(SEM_MATCH(
    'PREFIX : <http://www.example.org/electronics/>
```

```
     SELECT ?x ?cType ?p
     WHERE
      {?x :price ?p .
       ?x :cameraType ?cType .
      }
     ORDER BY ASC(?p)
     LIMIT 5
     OFFSET 5',
    SEM_Models('electronics'),
    SEM_Rulebases('RDFS'),
    null, null, null, ' ', null, null,
    'RDFUSER', 'NET1'))
ORDER BY SEM$ROWNUM;
```

Example 1-44 Query Using Full URIs

The SPARQL BASE keyword is used to set a global prefix. All relative IRIs will be resolved with the BASE IRI using the basic algorithm described in Section 5.2 of the *Uniform Resource Identifier (URI): Generic Syntax (RFC3986)* (`http://www.ietf.org/rfc/rfc3986.txt`). Example 1-44 is a simple query using full URIs, and Example 1-45 is an equivalent query using a base IRI.

```
SELECT *
  FROM TABLE(SEM_MATCH(
    'SELECT ?employee ?position
     WHERE
      {?x <http://www.example.org/employee> ?p .
       ?p <http://www.example.org/employee/name> ?employee .
       ?p <http://www.example.org/employee/position> ?pos .
       ?pos <http://www.example.org/positions/name> ?position
      }',
    SEM_Models('enterprise'),
    null,
     null, null, null, ' ', null, null,
   'RDFUSER', 'NET1'))
ORDER BY 1,2;
```

Example 1-45 Query Using a Base IRI

```
SELECT *
  FROM TABLE(SEM_MATCH(
    'BASE <http://www.example.org/>
     SELECT ?employee ?position
     WHERE
      {?x <employee> ?p .
       ?p <employee/name> ?employee .
       ?p <employee/position> ?pos .
       ?pos <positions/name> ?position
      }',
    SEM_Models('enterprise'),
    null,
    null, null, null, ' ', null, null,
   'RDFUSER', 'NET1'))
ORDER BY 1,2;
```

1.7.7 Graph Patterns: Support for SPARQL 1.1 Constructs

SEM_MATCH supports the following SPARQL 1.1 constructs:

- An expanded set of functions (all items in Table 1-14 in Graph Patterns: Support for Curly Brace Syntax_ and OPTIONAL_ FILTER_ UNION_ and GRAPH Keywords)

- Expressions in the SELECT Clause
- Subqueries
- Grouping and Aggregation
- Negation
- Value Assignment
- Property Paths

1.7.7.1 Expressions in the SELECT Clause

Expressions can be used in the SELECT clause to project the value of an expression from a query. A SELECT expression is composed of variables, RDF terms, arithmetic operators (+, -, *, /), Boolean operators and logical connectives (||, &&, !), comparison operators (<, >, <=, >=, =, !=), and any functions available for use in FILTER expressions. The expression must be aliased to a single variable using the AS keyword, and the overall *<expression> AS <alias>* fragment must be enclosed in parentheses. The alias variable cannot already be defined in the query. A SELECT expression may reference the result of a previous SELECT expression (that is, an expression that appears earlier in the SELECT clause).

Example 1-46 SPARQL SELECT Expression

Example 1-46 uses a SELECT expression to project the total price for each camera.

```
SELECT *
  FROM TABLE(SEM_MATCH(
    'PREFIX : <http://www.example.org/electronics/>
     SELECT ?x ((?p * (1-?d) * (1+?t)) AS ?totalPrice)
     WHERE
      {?x :price ?p .
       ?x :discount ?d .
       ?x :tax ?t .
       ?x :cameraType ?cType .
      }',
    SEM_Models('electronics'),
    SEM_Rulebases('RDFS'),
    null, null, null, ' ', null, null,
    'RDFUSER', 'NET1'));
```

Example 1-47 SPARQL SELECT Expressions (2)

Example 1-47 uses two SELECT expressions to project the discount price with and without sales tax.

```
SELECT *
  FROM TABLE(SEM_MATCH(
    'PREFIX : <http://www.example.org/electronics/>
     SELECT ?x ((?p * (1-?d)) AS ?preTaxPrice) ((?preTaxPrice * (1+?t)) AS ?finalPrice)
     WHERE
      {?x :price ?p .
       ?x :discount ?d .
       ?x :tax ?t .
       ?x :cameraType ?cType .
      }',
    SEM_Models('electronics'),
    SEM_Rulebases('RDFS'),
    null, null, null, ' ', null, null,
    'RDFUSER', 'NET1'));
```

1.7.7.2 Subqueries

Subqueries are allowed with SPARQL SELECT syntax. That is, fully-specified SPARQL SELECT queries may be embedded within other SPARQL SELECT queries. Subqueries have many uses, for example, limiting the number of results from a subcomponent of a query.

Example 1-48 SPARQL SELECT Subquery

Example 1-48 uses a subquery to find the manufacturer that makes the cheapest camera and then finds all other cameras made by this manufacturer.

```
SELECT *
  FROM TABLE(SEM_MATCH(
    'PREFIX : <http://www.example.org/electronics/>
     SELECT ?c1
     WHERE {?c1 rdf:type :Camera .
            ?c1 :manufacturer ?m .
            {
             SELECT ?m
             WHERE {?c2 rdf:Type :Camera .
                    ?c2 :price ?p .
                    ?c2 :manufacturer ?m .
             }
             ORDER BY ASC(?p)
             LIMIT 1
            }
     }',
    SEM_Models('electronics'),
    SEM_Rulebases('RDFS'),
    null, null, null, ' ', null, null,
    'RDFUSER', 'NET1'));
```

Subqueries are logically evaluated first, and the results are projected up to the outer query. Note that only variables projected in the subquery's SELECT clause are visible to the outer query.

1.7.7.3 Grouping and Aggregation

The GROUP BY keyword used to perform grouping. Syntactically, the GROUP BY keyword must appear after the WHERE clause and before any solution modifiers such as ORDER BY or LIMIT.

Aggregates are used to compute values across results within a group. An aggregate operates over a collection of values and produces a single value as a result. SEM_MATCH supports the following built-in Aggregates: COUNT, SUM, MIN, MAX, AVG, GROUP_CONCAT and SAMPLE. These aggregates are described in Table 1-17.

Table 1-17 Built-in Aggregates

Aggregate	Description
AVG(expression)	Returns the numeric average of *expression* over the values within a group.
COUNT(* | expression)	Counts the number of times *expression* has a bound, non-error value within a group; asterisk (*) counts the number of results within a group.

Table 1-17 (Cont.) Built-in Aggregates

Aggregate	Description
GROUP_CONCAT(expression [; SEPARATOR = "STRING"])	Performs string concatenation of *expression* over the values within a group. If provided, an optional separator string will be placed between each value.
MAX(expression)	Returns the maximum value of *expression* within a group based on the ordering defined by SPARQL ORDER BY.
MIN(expression)	Returns the minimum value of *expression* within a group based on the ordering defined by SPARQL ORDER BY.
SAMPLE(expression)	Returns *expression* evaluated for a single arbitrary value from a group.
SUM(expression)	Calculates the numeric sum of *expression* over the values within a group.

Certain restrictions on variable references apply when using grouping and aggregation. Only group-by variables (single variables in the GROUP BY clause) and alias variables from GROUP BY value assignments can be used in non-aggregate expressions in the SELECT or HAVING clauses.

Example 1-49 Simple Grouping Query

Example 1-49 shows a query that uses the GROUP BY keyword to find all the different types of cameras.

```
SELECT *
  FROM TABLE(SEM_MATCH(
    'PREFIX : <http://www.example.org/electronics/>
     SELECT ?cType
     WHERE
      {?x rdf:type :Camera .
       ?x :cameraType ?cType .
      }
     GROUP BY ?cType',
    SEM_Models('electronics'),
    SEM_Rulebases('RDFS'),
     null, null, null, ' ', null, null,
    'RDFUSER', 'NET1'));
```

A grouping query partitions the query results into a collection of groups based on a grouping expression (`?cType` in Example 1-49) such that each result within a group has the same values for the grouping expression. The final result of the grouping operation will include one row for each group.

Example 1-50 Complex Grouping Expression

A grouping expression consists of a sequence of one or more of the following: a variable, an expression, or a value assignment of the form (*<expression>* `as` *<alias>*). Example 1-50 shows a grouping query that uses one of each type of component in the grouping expression.

```
SELECT *
  FROM TABLE(SEM_MATCH(
    'PREFIX : <http://www.example.org/electronics/>
     SELECT ?cType ?totalPrice
     WHERE
      {?x rdf:type :Camera .
       ?x :cameraType ?cType .
       ?x :manufacturer ?m .
```

```
      ?x :price ?p .
      ?x :tax ?t .
     }
    GROUP BY ?cType (STR(?m)) ((?p*(1+?t)) AS ?totalPrice)',
   SEM_Models('electronics'),
   SEM_Rulebases('RDFS'),
   null, null, null, ' ', null, null,
   'RDFUSER', 'NET1'));
```

Example 1-51 Aggregation

Example 1-51 uses aggregates to select the maximum, minimum, and average price for each type of camera.

```
SELECT *
  FROM TABLE(SEM_MATCH(
   'PREFIX : <http://www.example.org/electronics/>
    SELECT ?cType
           (MAX(?p) AS ?maxPrice)
           (MIN(?p) AS ?minPrice)
           (AVG(?p) AS ?avgPrice)
    WHERE
     {?x rdf:type :Camera .
      ?x :cameraType ?cType .
      ?x :manufacturer ?m .
      ?x :price ?p .
     }
    GROUP BY ?cType',
   SEM_Models('electronics'),
   SEM_Rulebases('RDFS'),
   null, null, null, ' ', null, null,
   'RDFUSER', 'NET1'));
```

Example 1-52 Aggregation Without Grouping

If an aggregate is used without a grouping expression, then the entire result set is treated as a single group. Example 1-52 computes the total number of cameras for the whole data set.

```
SELECT *
  FROM TABLE(SEM_MATCH(
   'PREFIX : <http://www.example.org/electronics/>
    SELECT (COUNT(?x) as ?cameraCnt)
    WHERE
     { ?x rdf:type :Camera
     }',
   SEM_Models('electronics'),
   SEM_Rulebases('RDFS'),
   null, null, null, ' ', null, null,
   'RDFUSER', 'NET1'));
```

Example 1-53 Aggregation with DISTINCT

The DISTINCT keyword can optionally be used as a modifier for each aggregate. When DISTINCT is used, duplicate values are removed from each group before computing the aggregate. Syntactically, DISTINCT must appear as the first argument to the aggregate. Example 1-53 uses DISTINCT to find the number of distinct camera manufacturers. In this case, duplicate values of `STR(?m)` are removed before counting.

```
SELECT *
  FROM TABLE(SEM_MATCH(
   'PREFIX : <http://www.example.org/electronics/>
    SELECT (COUNT(DISTINCT STR(?m)) as ?mCnt)
```

```
     WHERE
      { ?x rdf:type :Camera .
        ?x :manufacturer ?m
      }',
    SEM_Models('electronics'),
    SEM_Rulebases('RDFS'),
     null, null, null, ' ', null, null,
    'RDFUSER', 'NET1'));
```

Example 1-54   HAVING Clause

The HAVING keyword can be used to filter groups based on constraints. HAVING expressions can be composed of variables, RDF terms, arithmetic operators (+, -, *, /), Boolean operators and logical connectives (||, &&, !), comparison operators (<, >, <=, >=, =, !=), aggregates, and any functions available for use in FILTER expressions. Syntactically, the HAVING keyword appears after the GROUP BY clause and before any other solution modifiers such as ORDER BY or LIMIT.

Example 1-54 uses a HAVING expression to find all manufacturers that sell cameras for less than $200.

```
SELECT *
  FROM TABLE(SEM_MATCH(
    'PREFIX : <http://www.example.org/electronics/>
     SELECT ?m
     WHERE
      { ?x rdf:type :Camera .
        ?x :manufacturer ?m .
        ?x :price ?p
      }
     GROUP BY ?m
     HAVING (MIN(?p) < 200)
     ORDER BY ASC(?m)',
    SEM_Models('electronics'),
    SEM_Rulebases('RDFS'),
    null, null, null, ' ', null, null,
    'RDFUSER', 'NET1'));
```

1.7.7.4 Negation

SEM_MATCH supports two forms of negation in SPARQL query patterns: NOT EXISTS and MINUS. NOT EXISTS can be used to filter results based on whether or not a graph pattern matches, and MINUS can be used to remove solutions based on their relation to another graph pattern.

Example 1-55   Negation with NOT EXISTS

Example 1-55 uses a NOT EXISTS FILTER to select those cameras that do not have any user reviews.

```
SELECT *
  FROM TABLE(SEM_MATCH(
    'PREFIX : <http://www.example.org/electronics/>
     SELECT ?x ?cType ?p
     WHERE
      {?x :price ?p .
       ?x :cameraType ?cType .
       FILTER( NOT EXISTS({?x :userReview ?r}) )
      }',
    SEM_Models('electronics'),
    SEM_Rulebases('RDFS'),
```

```
    null, null, null, ' ', null, null,
    'RDFUSER', 'NET1'));
```

Example 1-56 EXISTS

Conversely, the EXISTS operator can be used to ensure that a pattern matches. Example 1-56 uses an EXISTS FILTER to select only those cameras that have a user review.

```
SELECT *
  FROM TABLE(SEM_MATCH(
    'PREFIX : <http://www.example.org/electronics/>
     SELECT ?x ?cType ?p
     WHERE
      {?x :price ?p .
       ?x :cameraType ?cType .
       FILTER( EXISTS({?x :userReview ?r}) )
      }',
    SEM_Models('electronics'),
    SEM_Rulebases('RDFS'),
    null, null, null, ' ', null, null,
    RDFUSER', 'NET1'));
```

Example 1-57 Negation with MINUS

Example 1-57 uses MINUS to arrive at the same result as Example 1-55. Only those solutions that are not compatible with solutions from the MINUS pattern are included in the result. That is, if a solution has the same values for all shared variables as a solution from the MINUS pattern, it is removed from the result.

```
SELECT *
  FROM TABLE(SEM_MATCH(
    'PREFIX : <http://www.example.org/electronics/>
     SELECT ?x ?cType ?p
     WHERE
      {?x :price ?p .
       ?x :cameraType ?cType .
       MINUS {?x :userReview ?r}
      }',
    SEM_Models('electronics'),
    SEM_Rulebases('RDFS'),
    null, null, null, ' ', null, null,
    'RDFUSER', 'NET1'));
```

Example 1-58 Negation with NOT EXISTS (2)

NOT EXISTS and MINUS represent two different styles of negation and have different results in certain cases. One such case occurs when no variables are shared between the negation pattern and the rest of the query. For example, the NOT EXISTS query in Example 1-58 removes all solutions because `{?subj ?prop ?obj}` matches any triple, but the MINUS query in Example 1-59 removes no solutions because there are no shared variables.

```
SELECT *
  FROM TABLE(SEM_MATCH(
    'PREFIX : <http://www.example.org/electronics/>
     SELECT ?x ?cType ?p
     WHERE
      {?x :price ?p .
       ?x :cameraType ?cType .
       FILTER( NOT EXISTS({?subj ?prop ?obj}) )
      }',
    SEM_Models('electronics'),
    SEM_Rulebases('RDFS'),
```

```
    null, null, null, ' ', null, null,
    'RDFUSER', 'NET1'));
```

Example 1-59 Negation with MINUS (2)

```
SELECT *
  FROM TABLE(SEM_MATCH(
    'PREFIX : <http://www.example.org/electronics/>
     SELECT ?x ?cType ?p
     WHERE
      {?x :price ?p .
       ?x :cameraType ?cType .
       MINUS {?subj ?prop ?obj}
      }',
    SEM_Models('electronics'),
    SEM_Rulebases('RDFS'),
    null, null, null, ' ', null, null,
    'RDFUSER', 'NET1'));
```

1.7.7.5 Value Assignment

SEM_MATCH provides a variety of ways to assign values to variables in a SPARQL query.

The value of an expression can be assigned to a new variable in three ways: (1) expressions in the SELECT clause, (2) expressions in the GROUP BY clause, and (3) the BIND keyword. In each case, the new variable must not already be defined in the query. After assignment, the new variable can be used in the query and returned in results. As discussed in Expressions in the SELECT Clause, the syntax for value assignment is (<*expression*> AS <*alias*>) where *alias* is the new variable, for example, `((?price * (1+?tax)) AS ?totalPrice)`.

Example 1-60 Nested SELECT Expression

Example 1-60 uses a nested SELECT expression to compute the total price of a camera and assign the value to a variable (`?totalPrice`). This variable is then used in a FILTER in the outer query to find cameras costing less than $200.

```
SELECT *
 FROM TABLE(SEM_MATCH(
   'PREFIX : <http://www.example.org/electronics/>
    SELECT ?x ?cType ?totalPrice
    WHERE
     {?x :cameraType ?cType .
       { SELECT ?x ( ((?price*(1+?tax)) AS ?totalPrice )
         WHERE { ?x :price ?price .
                 ?x :tax ?tax }
       }
      FILTER (?totalPrice < 200)
     }',
   SEM_Models('electronics'),
   SEM_Rulebases('RDFS'),
   null, null, null, ' ', null, null,
   'RDFUSER', 'NET1'));
```

Example 1-61 BIND

The BIND keyword can be used inside a basic graph pattern to assign a value and is syntactically more compact than an equivalent nested SELECT expression. Example 1-61 uses the BIND keyword to expresses a query that is logically equivalent to Example 1-60.

```
SELECT *
  FROM TABLE(SEM_MATCH(
    'PREFIX : <http://www.example.org/electronics/>
```

```
    SELECT ?x ?cType ?totalPrice
    WHERE
     {?x :cameraType ?cType .
      ?x :price ?price .
      ?x :tax ?tax .
      BIND ( ((?price*(1+?tax)) AS ?totalPrice )
      FILTER (?totalPrice < 200)
     }',
   SEM_Models('electronics'),
   SEM_Rulebases('RDFS'),
   null, null, null, ' ', null, null,
  'RDFUSER', 'NET1'));
```

Example 1-62 GROUP BY Expression

Value assignments in the GROUP BY clause can subsequently be used in the SELECT clause, the HAVING clause, and the outer query (in the case of a nested grouping query). Example 1-62 uses a GROUP BY expression to find the maximum number of megapixels for cameras at each price point less than $1000.

```
SELECT *
  FROM TABLE(SEM_MATCH(
    'PREFIX : <http://www.example.org/electronics/>
     SELECT ?totalPrice (MAX(?mp) as ?maxMP)
     WHERE
      {?x rdf:type :Camera .
       ?x :price ?price .
       ?x :tax ?tax .
       GROUP BY ( ((?price*(1+?tax)) AS ?totalPrice )
       HAVING (?totalPrice < 1000)
      }',
    SEM_Models('electronics'),
    SEM_Rulebases('RDFS'),
    null, null));
```

Example 1-63 VALUES

In addition to the preceding three ways to assign the value of an expression to a new variable, the VALUES keyword can be used to introduce an unordered solution sequence that is combined with the query results through a join operation. A VALUES block can appear inside a query pattern or at the end of a SPARQL SELECT query block after any solution modifiers. The VALUES construct can be used in subqueries.

Example 1-63 uses the VALUES keyword to constrain the query results to DSLR cameras made by `:Company1` or any type of camera made by `:Company2`. The keyword UNDEF is used to represent an unbound variable in the solution sequence.

```
SELECT *
  FROM TABLE(SEM_MATCH(
    'PREFIX : <http://www.example.org/electronics/>
     SELECT ?x ?cType ?m
     WHERE
      { ?x rdf:type :Camera .
        ?x :cameraType ?cType .
        ?x :manufacturer ?m
      }
     VALUES (?cType ?m)
     { ("DSLR" :Company1)
       (UNDEF  :Company2)
     }',
    SEM_Models('electronics'),
    SEM_Rulebases('RDFS'),
```

```
    null, null, null, ' ', null, null,
    'RDFUSER', 'NET1'));
```

Example 1-64 Simplified VALUES Syntax

A simplified syntax can be used for the common case of a single variable. Specifically, the parentheses around the variable and each solution can be omitted. Example 1-64 uses the simplified syntax to constrain the query results to cameras made by `:Company1` or `:Company2`.

```
SELECT *
  FROM TABLE(SEM_MATCH(
    'PREFIX : <http://www.example.org/electronics/>
     SELECT ?x ?cType ?m
     WHERE
      { ?x rdf:type :Camera .
        ?x :cameraType ?cType .
        ?x :manufacturer ?m
      }
     VALUES ?m
     { :Company1
       :Company2
     }',
    SEM_Models('electronics'),
    SEM_Rulebases('RDFS'),
    null, null, null, ' ', null, null,
    'RDFUSER', 'NET1'));
```

Example 1-65 Inline VALUES Block

Example 1-65 also constrains the query results to any camera made by `:Company1` or `:Company2`, but specifies the VALUES block inside the query pattern.

```
SELECT *
  FROM TABLE(SEM_MATCH(
    'PREFIX : <http://www.example.org/electronics/>
     SELECT ?x ?cType ?m
     WHERE
      { VALUES ?m { :Company1 :Company2 }
        ?x rdf:type :Camera .
        ?x :cameraType ?cType .
        ?x :manufacturer ?m
      }',
    SEM_Models('electronics'),
    SEM_Rulebases('RDFS'),
    null, null, null, ' ', null, null,
    'RDFUSER', 'NET1'));
```

1.7.7.6 Property Paths

A SPARQL Property Path describes a possible path between two RDF resources (nodes) in an RDF graph. A property path appears in the predicate position of a triple pattern and uses a regular expression-like syntax to place constraints on the properties (edges) making up a path from the subject of the triple pattern to the object of a triple pattern. Property paths allow SPARQL queries to match arbitrary length paths in the RDF graph and also provide a more concise way to express other graph patterns.

Table 1-18 describes the syntax constructs available for constructing SPARQL Property Paths. Note that *iri* is either an IRI or a prefixed name, and *elt* is a property path element, which may itself be composed of other property path elements.

Table 1-18 Property Path Syntax Constructs

Syntax Construct	Matches
iri	An IRI or a prefixed name. A path of length 1 (one).
^elt	Inverse path (object to subject).
!iri or *!(iri1 | … | irin)*	Negated property set. An IRI that is not one of *irii*.
!^iri or *!(iri1 | … | irij | ^irij+1 | … | ^irin)*	Negated property set with some inverse properties. An IRI that is not one of *irii*, nor one of *irij+1...irin* as reverse paths. *!^iri* is short for *!(^iri)*. The order of properties and inverse properties is not important. They can occur in mixed order.
(elt)	A group path *elt*; brackets control precedence.
elt_1 / elt_2	A sequence path of elt_1, followed by elt_2.
elt_1 | elt_2	An alternative path of elt_1, or elt_2 (all possibilities are tried).
*elt**	A path of zero or more occurrences of *elt*.
elt+	A path of one or more occurrences of *elt*.
elt?	A path of zero or one occurrence of *elt*.

The precedence of the syntax constructs is as follows (from highest to lowest):

- IRI, prefixed names
- Negated property sets
- Groups
- Unary operators *, ?, +
- Unary ^ inverse links
- Binary operator /
- Binary operator |

Precedence is left-to-right within groups.

Special Considerations for Property Path Operators + and *

In general, truly unbounded graph traversals using the + (plus sign) and * (asterisk) operator can be very expensive. For this reason, a depth-limited version of the + and * operator is used by default, and the default depth limit is 10. In addition, the depth-limited implementation can be run in parallel. The `ALL_MAX_PP_DEPTH(n)` SEM_MATCH query option or the `MAX_PP_DEPTH(n)` inline HINT0 query optimizer hint can be used to change the depth-limit setting. To achieve a truly unbounded traversal, you can set a depth limit of less than 1 to fall back to a CONNECT BY-based implementation.

Query Hints for Property Paths

Other query hints are available to influence the performance of property path queries. The `ALLOW_PP_DUP=T` query option can be used with * and + queries to allow duplicate results. Allowing duplicate results may return the first rows from a query faster. In addition, `ALL_USE_PP_HASH` and `ALL_USE_PP_NL` query options are available to influence the join types used when evaluating property path expressions. Analogous `USE_PP_HASH` and `USE_PP_NL` inline HINT0 query optimizer hints can also be used.

Example 1-66 SPARQL Property Path (Using rdfs:subClassOf Relations)

Example 1-66 uses a property path to find all Males based on transitivity of the `rdfs:subClassOf relationship`. A property path allows matching an arbitrary number of consecutive `rdfs:subClassOf` relations.

```
SELECT x, name
  FROM TABLE(SEM_MATCH(
    '{ ?x foaf:name ?name .
       ?x rdf:type ?t .
       ?t rdfs:subClassOf* :Male }',
    SEM_Models('family'),
    null,
    SEM_ALIASES(SEM_ALIAS('','http://www.example.org/family/')
                SEM_ALIAS('foaf',' http://xmlns.com/foaf/0.1/')),
    null, null, ' ', null, null,
    'RDFUSER', 'NET1'));
```

Example 1-67 SPARQL Property Path (Using foaf:friendOf or foaf:knows Relationships)

Example 1-67 uses a property path to find all of Scott's close friends (those people reachable within two hops using `foaf:friendOf` or `foaf:knows` relationships).

```
SELECT name
  FROM TABLE(SEM_MATCH(
    '{ { :Scott (foaf:friendOf | foaf:knows) ?f }
       UNION
       { :Scott (foaf:friendOf | foaf:knows)/(foaf:friendOf | foaf:knows) ?f }
       ?f foaf:name ?name .
       FILTER (!sameTerm(?f, :Scott)) }',
    SEM_Models('family'),
    null,
    SEM_ALIASES(SEM_ALIAS('','http://www.example.org/family/'),
                SEM_ALIAS('foaf',' http://xmlns.com/foaf/0.1/')),
    null, null, ' ', null, null,
    'RDFUSER', 'NET1'));
```

Example 1-68 Specifying Property Path Maximum Depth Value

Example 1-68 specifies a maximum depth of 12 for all property path expressions with the `ALL_MAX_PP_DEPTH(n)` query option value.

```
SELECT x, name
  FROM TABLE(SEM_MATCH(
    '{ ?x foaf:name ?name .
       ?x rdf:type ?t .
       ?t rdfs:subClassOf* :Male }',
    SEM_Models('family'),
    null,
    SEM_ALIASES(SEM_ALIAS('','http://www.example.org/family/')
                SEM_ALIAS('foaf',' http://xmlns.com/foaf/0.1/')),
    null,
    null,
    ' ALL_MAX_PP_DEPTH(12) ',
    null, null,
    'RDFUSER', 'NET1'));
```

Example 1-69 Specifying Property Path Join Hint

Example 1-69 shows an inline HINT0 query optimizer hint that requests a nested loop join for evaluating the property path expression.

```
SELECT x, name
  FROM TABLE(SEM_MATCH(
    '{ # HINT0={ USE_PP_NL }
       ?x foaf:name ?name .
       ?x rdf:type ?t .
       ?t rdfs:subClassOf* :Male }',
    SEM_Models('family'),
    null,
    SEM_ALIASES(SEM_ALIAS('','http://www.example.org/family/')
                SEM_ALIAS('foaf',' http://xmlns.com/foaf/0.1/')),
    null, null, ' ', null, null,
    'RDFUSER', 'NET1'));
```

1.7.8 Graph Patterns: Support for SPARQL 1.1 Federated Query

SEM_MATCH supports SPARQL 1.1 Federated Query (see `http://www.w3.org/TR/sparql11-federated-query/#SPROT`). The SERVICE construct can be used to retrieve results from a specified SPARQL endpoint URL. With this capability, you can combine local RDF data (native RDF data or RDF views of relational data) with other, possibly remote, RDF data served by a W3C standards-compliant SPARQL endpoint.

Example 1-70 SPARQL SERVICE Clause to Retrieve All Triples

Example 1-70 shows a query that uses a SERVICE clause to retrieve all triples from the SPARQL endpoint available at `http://www.example1.org/sparql`.

```
SELECT s, p, o
  FROM TABLE(SEM_MATCH(
    'SELECT ?s ?p ?o
     WHERE {
       SERVICE <http://www.example1.org/sparql>{ ?s ?p ?o }
     }',
    SEM_Models('electronics'),
    null, null, null, null, ' ',
    null, null,
    'RDFUSER', 'NET1'));
```

Example 1-71 SPARQL SERVICE Clause to Join Remote and Local RDF Data

Example 1-71 joins remote RDF data with local RDF data. This example joins camera types `?cType` from local RDF graph `electronics` with the camera names `?name` from the SPARQL endpoint at `http://www.example1.org/sparql`.

```
SELECT cType, name
  FROM TABLE(SEM_MATCH(
    'PREFIX : <http://www.example.org/electronics/>
     SELECT ?cType ?name
     WHERE {
       ?s :cameraType ?cType
       SERVICE <http://www.example1.org/sparql>{ ?s :name ?name }
      }',
    SEM_Models('electronics'),
    null, null, null, null, ' ',
    null, null,
    'RDFUSER', 'NET1'));
```

- Privileges Required to Execute Federated SPARQL Queries
- SPARQL SERVICE Join Push Down
- SPARQL SERVICE SILENT

- Using a Proxy Server with SPARQL SERVICE
- Accessing SPARQL Endpoints with HTTP Basic Authentication

1.7.8.1 Privileges Required to Execute Federated SPARQL Queries

You need certain database privileges to use the SERVICE construct within SEM_MATCH queries. You should be granted EXECUTE privilege on the SPARQL_SERVICE function by a user with DBA privileges. The following example grants this access to a user named RDFUSER:

```
grant execute on sparql_service to rdfuser;
```

Also, an Access Control List (ACL) should be used to grant the CONNECT privilege to the user attempting a federated query. Example 1-72 creates a new ACL to grant the user RDFUSER the CONNECT privilege and assigns the domain * to the ACL. For more information about ACLs, see *Oracle Database PL/SQL Packages and Types Reference*.

Example 1-72 Access Control List and Host Assignment

```
dbms_network_acl_admin.create_acl (
  acl       => 'rdfuser.xml',
  description => 'Allow rdfuser to query SPARQL endpoints',
  principal => 'RDFUSER',
  is_grant  => true,
  privilege => 'connect'
);

dbms_network_acl_admin.assign_acl (
  acl  => 'rdfuser.xml',
  host => '*'
);
```

After the necessary privileges are granted, you are ready to execute federated queries from SEM_MATCH

1.7.8.2 SPARQL SERVICE Join Push Down

The SPARQL SERVICE Join Push Down (`SERVICE_JPDWN=T`) feature can be used to improve the performance of certain SPARQL SERVICE queries. By default, the query pattern within the SERVICE clause is executed first on the remote SPARQL endpoint. The full result of this remote execution is then joined with the local portion of the query. This strategy can result in poor performance if the local portion of the query is very selective and the remote portion of the query is very unselective.

The SPARQL SERVICE Join Push Down feature cannot be used in a query that contains more than one SERVICE clause.

Example 1-73 SPARQL SERVICE Join Push Down

Example 1-73 shows the SPARQL SERVICE Join Push Down feature.

```
SELECT s, prop, obj
  FROM TABLE(SEM_MATCH(
    'PREFIX : <http://www.example.org/electronics/>
     SELECT ?s ?prop ?obj
     WHERE {
       ?s rdf:type   :Camera .
       ?s :modelName "Camera 12345"
       SERVICE <http://www.example1.org/sparql> { ?s ?prop ?obj }
      }',
```

```
    SEM_Models('electronics'),
    null, null, null, null, ' SERVICE_JPDWN=T ',
    null, null,
    'RDFUSER', 'NET1'));
```

In Example 1-73, the local portion of the query will return a very small number of rows, but the remote portion of the query is completely unbound and will return the entire remote dataset. When the `SERVICE_JPDWN=T` option is specified, SEM_MATCH performs a nested-loop style evaluation by first executing the local portion of the query and then executing a modified version of the remote query once for each row returned by the local portion. The remote query is modified with a FILTER clause that effectively performs a substitution for the join variable `?s`. For example, if `<urn:camera1>` and `<urn:camera2>` are returned from the local portion of Example 1-73 as bindings for `?s`, then the following two queries are sent to the remote endpoint: `{ ?s ?prop ?obj FILTER (?s = <urn:camera1>) }` and `{ s ?prop ?obj FILTER (?s = <urn:camera2>) }`.

1.7.8.3 SPARQL SERVICE SILENT

When the SILENT keyword is used in federated queries, errors while accessing the specified remote SPARQL endpoint will be ignored. If the SERVICE SILENT request fails, a single solution with no bindings will be returned.

Example 1-74 uses SERVICE with the SILENT keyword inside an OPTIONAL clause, so that, when connection errors accessing `http://www.example1.org/sparql` appear, such errors will be ignored and all the rows retrieved from triple `?s :cameratype ?k` will be combined with a null value for `?n`.

Example 1-74 SPARQL SERVICE with SILENT Keyword

```
SELECT s, n
  FROM TABLE(SEM_MATCH(
    'PREFIX : <http://www.example.org/electronics/>
     SELECT ?s ?n
     WHERE {
       ?s :cameraType ?k
       OPTIONAL { SERVICE SILENT <http://www.example1.org/sparql>{ ?k :name ?n } }
      }',
    SEM_Models('electronics'),
    null, null, null, null, ' ', null, null,
    'RDFUSER', 'NET1'));
```

1.7.8.4 Using a Proxy Server with SPARQL SERVICE

The following methods are available for sending SPARQL SERVICE requests through an HTTP proxy:

- Specifying the HTTP proxy that should be used for requests in the current session. This can be done through the SET_PROXY function of UTL_HTTP package. Example 1-75 sets the proxy `proxy.example.com` to be used for HTTP requests, excluding those to hosts in the domain `example2.com`. (For more information about the SET_PROXY procedure, see *Oracle Database PL/SQL Packages and Types Reference*.)
- Using the SERVICE_PROXY SEM_MATCH option, which allows setting the proxy address for SPARQL SERVICE request. However, in this case no exceptions can be specified, and all requests are sent to the given proxy server. Example 1-76 shows a SEM_MATCH query where the proxy address `proxy.example.com` at port 80 is specified.

Example 1-75   Setting Proxy Server with UTL_HTTP.SET_PROXY

```
BEGIN
  UTL_HTTP.SET_PROXY('proxy.example.com:80', 'example2.com');
END;
/
```

Example 1-76   Setting Proxy Server in SPARQL SERVICE

```
SELECT *
  FROM TABLE(SEM_MATCH(
    'SELECT *
     WHERE {
       SERVICE <http://www.example1.org/sparql>{ ?s ?p ?o }
      }',
    SEM_Models('electronics'),
    null, null, null, null, ' SERVICE_PROXY=proxy.example.com:80 ',
    null, null,
    'RDFUSER', 'NET1'));
```

1.7.8.5 Accessing SPARQL Endpoints with HTTP Basic Authentication

To allow accessing of SPARQL endpoints with HTTP Basic Authentication, user credentials should be saved in Session Context SDO_SEM_HTTP_CTX. A user with DBA privileges must grant EXECUTE on this context to the user that wishes to use basic authentication. The following example grants this access to a user named RDFUSER:

```
grant execute on mdsys.sdo_sem_http_ctx to rdfuser;
```

After the privilege is granted, the user should save the user name and password for each SPARQL Endpoint with HTTP Authentication through functions `mdsys.sdo_sem_http_ctx.set_usr` and `mdsys.sdo_sem_http_ctx.set_pwd`. The following example sets a user name and password for the SPARQL endpoint at `http://www.example1.org/sparql`:

```
BEGIN
  mdsys.sdo_sem_http_ctx.set_usr('http://www.example1.org/sparql','user');
  mdsys.sdo_sem_http_ctx.set_pwd('http://www.example1.org/sparql','pwrd');
END;
/
```

1.7.9 Inline Query Optimizer Hints

In SEM_MATCH, the SPARQL comment construct has been overloaded to allow inline HINT0 query optimizer hints. In SPARQL, the hash (#) character indicates that the remainder of the line is a comment. To associate an inline hint with a particular BGP, place a HINT0 hint string inside a SPARQL comment and insert the comment between the opening curly bracket ({) and the first triple pattern in the BGP. Inline hints enable you to influence the execution plan for each BGP in a query.

Inline optimizer hints override any hints passed to SEM_MATCH through the options argument. For example, a global ALL_ORDERED hint applies to each BGP that does not specify an inline optimizer hint, but those BGPs with an inline hint use the inline hint instead of the ALL_ORDERED hint.

Example 1-77   Inline Query Optimizer Hints (BGP_JOIN)

The following example shows a query with inline query optimizer hints.

```
SELECT x, y, hp, cp
  FROM TABLE(SEM_MATCH(
    '{ # HINT0={ LEADING(t0) USE_NL(?x ?y ?bd) }
      ?x :grandParentOf ?y . ?x rdf:type :Male . ?x :birthDate ?bd
      OPTIONAL { # HINT0={ LEADING(t0 t1) BGP_JOIN(USE_HASH) }
                 ?x :homepage ?hp . ?x :cellPhoneNum ?cp }
     }',
    SEM_Models('family'),
    SEM_Rulebases('RDFS','family_rb'),
    SEM_ALIASES(SEM_ALIAS('','http://www.example.org/family/')),
    null, null, ' ', null, null,
    'RDFUSER', 'NET1'));
```

The BGP_JOIN hint influences inter-BGP joins and has the following syntax: `BGP_JOIN(<join_type>)`, where *<join_type>* is USE_HASH or USE_NL. Example 1-77 uses the BGP_JOIN(USE_HASH) hint to specify that a hash join should be used when joining the OPTIONAL BGP with its parent BGP.

Inline optimizer hints override any hints passed to SEM_MATCH through the `options` argument. For example, a global ALL_ORDERED hint applies to each BGP that does not specify an inline optimizer hint, but those BGPs with an inline hint use the inline hint instead of the ALL_ORDERED hint.

Example 1-78 Inline Query Optimizer Hints (ANTI_JOIN)

The ANTI_JOIN hint influences the evaluation of NOT EXISTS and MINUS clauses. This hint has the syntax `ANTI_JOIN(<join_type>)`, where <join_type> is HASH_AJ, NL_AJ, or MERGE_AJ. The following example uses a hint to indicate that a hash anti join should be used. Global ALL_AJ_HASH, ALL_AJ_NL, ALL_AJ_MERGE can be used in the options argument of SEM_MATCH to influence the join type of all NOT EXISTS and MINUS clauses in the entire query.

```
SELECT x, y
  FROM TABLE(SEM_MATCH(
    'SELECT ?x ?y
     WHERE {
       ?x :grandParentOf ?y . ?x rdf:type :Male . ?x :birthDate ?bd
       FILTER (
         NOT EXISTS {# HINT0={ ANTI_JOIN(HASH_AJ) }
                     ?x :homepage ?hp . ?x :cellPhoneNum ?cp })
     }',
    SEM_Models('family'),
    SEM_Rulebases('RDFS','family_rb'),
    SEM_ALIASES(SEM_ALIAS('','http://www.example.org/family/')),
    null, null, ' ', null, null,
    'RDFUSER', 'NET1'));
```

Example 1-79 Inline Query Optimizer Hints (NON_NULL)

`HINT0={ NON_NULL}` is supported in SPARQL SELECT clauses to signify that a particular variable is always bound (that is, has a non-null value in each result row). This hint allows the query compiler to optimize joins for values produced by SELECT expressions. These optimizations cannot be applied by default because it cannot be guaranteed that expressions will produce non-null values for all possible input. If you know that a SELECT expression will not produce any null values for a particular query, using this NON_NULL hint can significantly increase performance. This hint should be specified in the comment in a line before the 'AS' keyword of a SELECT expression.

The following example shows the NON_NULL hint option used in a SEM_MATCH query, specifying that variable `?full_name` is definitely bound.

```
SELECT s, t
  FROM TABLE(SEM_MATCH(
    'SELECT * WHERE {
       ?s :name ?full_name
       { SELECT (CONCAT(?fname, " ", ?lname) # HINT0={ NON_NULL }
                  AS ?full_name)
         WHERE {
           ?t :fname ?fname .
           ?t :lname ?lname }
         }
     }',
    SEM_Models('family'),
    SEM_Rulebases('RDFS','family_rb'),
    SEM_ALIASES(SEM_ALIAS('','http://www.example.org/family/')),
    null, null, ' ', null, null,
    'RDFUSER', 'NET1'));
```

1.7.10 Full-Text Search

The Oracle-specific `orardf:textContains` SPARQL FILTER function uses full-text indexes on the RDF_VALUE$ table. This function has the following syntax (where `orardf` is a built-in prefix that expands to `<http://xmlns.oracle.com/rdf/>`):

```
orardf:textContains(variable, pattern)
```

The first argument to `orardf:textContains` must be a local variable (that is, a variable present in the BGP that contains the `orardf:textContains` filter), and the second argument must be a constant plain literal.

For example, `orardf:textContains(x, y)` returns `true` if `x` matches the expression `y`, where `y` is a valid expression for the Oracle Text SQL operator CONTAINS. For more information about such expressions, see *Oracle Text Reference*.

Before using `orardf:textContains`, you must create an Oracle Text index for the RDF network. To create such an index, invoke the SEM_APIS.ADD_DATATYPE_INDEX procedure as follows:

```
EXECUTE SEM_APIS.ADD_DATATYPE_INDEX('http://xmlns.oracle.com/rdf/text',
network_owner=>'RDFUSER', network_name=>'NET1');
```

Performance for wildcard searches like `orardf:textContains(?x, "%abc%")` can be improved by using prefix and substring indexes. You can include any of the following options to the SEM_APIS.ADD_DATATYPE_INDEX procedure:

- `PREFIX_INDEX=TRUE` – for adding prefix index
- `PREFIX_MIN_LENGTH=<number>` – minimum length for prefix index tokens
- `PREFIX_MAX_LENGTH=<number>` – maximum length for prefix index tokens
- `SUBSTRING_INDEX=TRUE` – for adding substring index
- `LOGGING=T` – to enable logging for text index

For more information about Oracle Text indexing elements, see *Oracle Text Reference*.

When performing large bulk loads into a RDF network with a text index, the overall load time may be faster if you drop the text index, perform the bulk load, and then re-create the text index. See Using Data Type Indexes for more information about data type indexing.

After creating a text index, you can use the `orardf:textContains` FILTER function in SEM_MATCH queries. Example 1-80 uses `orardf:textContains` to find all grandfathers whose names start with the letter *A* or *B*.

Example 1-80 Full-Text Search

```
SELECT x, y, n
  FROM TABLE(SEM_MATCH(
    'PREFIX : <http://www.example.org/family/>
     SELECT *
     WHERE {
       ?x :grandParentOf ?y . ?x rdf:type :Male . ?x :name ?n
       FILTER (orardf:textContains(?n, " A% | B% ")) }',
    SEM_Models('family'),
    SEM_Rulebases('RDFS','family_rb'),
    null, null, null, ' ', null, null,
   'RDFUSER', 'NET1'));
```

Example 1-81 orardf:textScore

The ancillary operator `orardf:textScore` can be used in combination with `orardf:textContains` to rank results by the goodness of their text match. There are, however, limitations when using `orardf:textScore`. The `orardf:textScore` invocation must appear as a SELECT expression in the SELECT clause immediately surrounding the basic graph pattern that contains the corresponding `orardf:textContains` FILTER. The alias for this SELECT expression can then be used in other parts of the query. In addition, a `REWRITE=F'` query hint must be used in the `options` argument of SEM_MATCH.

The following example finds text matches with score greater than 0.5. Notice that an additional invocation id argument is required for `orardf:textContains`, so that it can be linked to the `orardf:textScore` invocation with the same invocation id. The invocation ID is an arbitrary integer constant used to match a primary operator with its ancillary operator.

```
SELECT x, y, n, scr
  FROM TABLE(SEM_MATCH(
    'PREFIX <http://www.example.org/family/>
     SELECT *
     WHERE {
       { SELECT ?x ?y ?n (orardf:textScore(123) AS ?scr)
         WHERE {
           ?x :grandParentOf ?y . ?x rdf:type :Male . ?x :name ?n
           FILTER (orardf:textContains(?n, " A% | B% ", 123)) }
       }
       FILTER (?scr > 0.5)
     }',
    SEM_Models('family'),
    SEM_Rulebases('RDFS','family_rb'),
    null,
    null,
    null,
    ' REWRITE=F ',
    null, null,
    'RDFUSER', 'NET1'));
```

Example 1-82 orardf:like

For a lightweight text search, you can use the `orardf:like` function, which performs simple test for pattern matching using the Oracle SQL operator `LIKE`. The `orardf:like` function has the following syntax:

- `orardf:like(string, pattern)`
- `orardf:like(string, pattern, flags)`

The first argument of `orardf:like` can be any variable or RDF term, as opposed to `orardf:Contains`, which requires the first argument to be a local variable. When the first argument to `orardf:like` is a URI, the match is performed against the URI suffix only. The second argument must be a pattern expression, which can contain the following special pattern-matching characters:

- The percent sign (%) can match zero or more characters.
- The underscore (_) matches exactly one character.

The flags argument must be a constant string. The flag "`i`" is supported to allow a case-insensitive search.

The following example shows a percent sign (%) wildcard search to find all grandparents whose URIs start with `Ja`.

```
SELECT x, y, n
  FROM TABLE(SEM_MATCH(
    'PREFIX : <http://www.example.org/family/>
     SELECT *
     WHERE {
        ?x :grandParentOf ?y . ?y :name ?n
        FILTER (orardf:like(?x, "Ja%")) }',
    SEM_Models('family'),
    SEM_Rulebases('RDFS','family_rb'),

    null, null, null, ' ', null, null,
    'RDFUSER', 'NET1'));
```

The following example shows an underscore (_) wildcard search to find all the grandchildren whose names start with `J` followed by two characters and end with `k`. The case-insensitive flag "`i`" is used to make the search case-insensitive.

```
SELECT x, y, n
  FROM TABLE(SEM_MATCH(
    'PREFIX : <http://www.example.org/family/>
     SELECT *
     WHERE {
        ?x :grandParentOf ?y . ?y :name ?n
        FILTER (orardf:like(?n, "j__k", "i"))
    }',
    SEM_Models('family'),
    SEM_Rulebases('RDFS','family_rb'),

    null, null, null, ' ', null, null,
    'RDFUSER', 'NET1'));
```

For efficient execution of `orardf:like`, you can create an index using the SEM_APIS.ADD_DATATYPE_INDEX procedure with `http://xmlns.oracle.com/rdf/like` as the data type URI. This index can speed up queries when the first argument is a local variable and the leading character of the search pattern is not a wildcard. The underlying index is a simple function-based B-Tree index on a varchar function, which has lower maintenance and storage costs than a full Oracle Text index. The index for `orardf:like` is created as follows:

```
EXECUTE SEM_APIS.ADD_DATATYPE_INDEX('http://xmlns.oracle.com/rdf/like');
```

1.7.11 Spatial Support

RDF Graph supports storage and querying of spatial geometry data through the OGC GeoSPARQL standard and through Oracle-specific SPARQL extensions. Geometry data can be stored as `orageo:WKTLiteral`, `ogc:wktLiteral`, `ogc:gmlLiteral`, `ogc:geoJSONLiteral`, or `ogc:kmlLiteral` typed literals, and geometry data can be queried using several query functions for spatial operations. Spatial indexing for increased performance is also supported.

`orageo` is a built-in prefix that expands to `<http://xmlns.oracle.com/rdf/geo/>`, `ogc` is a built-in prefix that expands to `<http://www.opengis.net/ont/geosparql#>`, and `ogcf` is a built-in prefix that expands to `<http://www.opengis.net/def/function/geosparql>`.

- OGC GeoSPARQL Support
- Representing Spatial Data in RDF
- Validating Geometries
- Indexing Spatial Data
- Querying Spatial Data
- Using Long Literals with GeoSPARQL Queries

1.7.11.1 OGC GeoSPARQL Support

RDF Graph supports the following conformance classes for the OGC GeoSPARQL standard (`http://www.opengeospatial.org/standards/geosparql`) using well-known text (WKT) serialization and the Simple Features relation family.

- Core
- Topology Vocabulary Extension (*Simple Features*)
- Geometry Extension (*WKT, 1.2.0*)
- Geometry Topology Extension (*Simple Features, WKT, 1.2.0*)
- RDFS Entailment Extension (*Simple Features, WKT, 1.2.0*)

RDF Graph supports the following conformance classes for OGC GeoSPARQL using Geography Markup Language (GML) serialization and the Simple Features relation family.

- Core
- Topology Vocabulary Extension (Simple Features)
- Geometry Extension (GML, 3.1.1)
- Geometry Topology Extension (Simple Features, GML, 3.1.1)
- RDFS Entailment Extension (Simple Features, GML, 3.1.1)

RDF Graph supports the following conformance classes for OGC GeoSPARQL using Geographic JavaScript Object Notation (GeoJSON) serialization and the Simple Features relation family.

- Core
- Topology Vocabulary Extension (Simple Features)
- Geometry Extension (GeoJSON, 1.0)
- Geometry Topology Extension (Simple Features, GeoJSON, 1.0)
- RDFS Entailment Extension (Simple Features, GeoJSON, 1.0)

RDF Graph supports the following conformance classes for OGC GeoSPARQL using Keyhole Markup Language (KML) serialization and the Simple Features relation family.

- Core
- Topology Vocabulary Extension (Simple Features)
- Geometry Extension (KML, 2.1)
- Geometry Topology Extension (Simple Features, KML, 2.1)
- RDFS Entailment Extension (Simple Features, KML, 2.1)

Specifics for representing and querying spatial data using GeoSPARQL are covered in sections that follow this one.

1.7.11.2 Representing Spatial Data in RDF

Spatial geometries can be represented in RDF as `orageo:WKTLiteral`, `ogc:wktLiteral`, `ogc:gmlLiteral`, `ogc:geoJSONLiteral`, or `ogc:kmlLiteral` typed literals. In this document, the term **geometry literal** is used to refer to an RDF literal that is any one of these five literal types.

Example 1-83 Spatial Point Geometry Represented as orageo:WKTLiteral

The following example shows the `orageo:WKTLiteral` encoding for a simple point geometry.

```
"Point(-83.4 34.3)"^^<http://xmlns.oracle.com/rdf/geo/WKTLiteral>
```

Example 1-84 Spatial Point Geometry Represented as ogc:wktLiteral

The following example shows the `ogc:wktLiteral` encoding for the same point as in the preceding example.

```
"Point(-83.4 34.3)"^^<http://www.opengis.net/ont/geosparql#wktLiteral>
```

Both `orageo:WKTLiteral` and `ogc:wktLiteral` encodings consist of an optional spatial reference system URI, followed by a Well-Known Text (WKT) string that encodes a geometry value. The spatial reference system URI and the WKT string should be separated by a whitespace character.

Supported spatial reference system URIs have the following form `<http://www.opengis.net/def/crs/EPSG/0/{srid}>`, where `{srid}` is a valid spatial reference system ID defined by the European Petroleum Survey Group (EPSG). For URIs that are not in the EPSG Geodetic Parameter Dataset, the spatial reference system URIs used have the form `<http://xmlns.oracle.com/rdf/geo/srid/{srid}>`, where `{srid}` is a valid spatial reference system ID from Oracle Spatial. If a geometry literal value does not include a spatial reference system URI, then the default spatial reference system, WGS84 Longitude-Latitude (URI `<http://www.opengis.net/def/crs/OGC/1.3/CRS84>`), is used. The same default spatial reference system is used when geometry literal values are encountered in a query string.

Example 1-85 Spatial Point Geometry Represented as ogc:gmlLiteral

The following example shows the `ogc:gmlLiteral` encoding for a point geometry.

```
"<gml:Point srsName=\"urn:ogc:def:crs:EPSG::8307\" xmlns:gml=\"http://
www.opengis.net/gml\"><gml:posList srsDimension=\"2\">-83.4 34.3</
gml:posList></gml:Point>"^^<http://www.opengis.net/ont/geosparql#gmlLiteral>
```

`ogc:gmlLiteral` encodings consist of a valid element from the GML schema that implements a subtype of GM_Object. In contrast to WKT literals, A GML encoding explicitly includes spatial reference system information, so a spatial reference system URI prefix is not needed.

Example 1-86 Spatial Polygon Geometry Represented as ogc:geoJSONLiteral

The following example shows a valid `ogc:geoJSONLiteral` encoding for a polygon geometry.

```
"{ \"type\": \"Polygon\", \"coordinates\": [ [ [-75, 44], [-75, 42], [-72,
42],
[-72, 45], [-74, 45], [-75, 44] ] ] }"^^<http://www.opengis.net/ont/
geosparql#geoJSONLiteral>
```

`ogc:geoJSONLiteral` encodings consist of a valid GeoJSON serialization of a geometry object. `ogc:geoJSONLiterals` are always interpreted using WGS84 geodetic longitude-latitude spatial reference system.

Example 1-87 Spatial Polygon Geometry Represented as ogc:kmlLiteral

The following example shows the `ogc:kmlLiteral` encoding for a polygon geometry.

```
"<Polygon><extrude>0</extrude><tessellate>0</
tessellate><altitudeMode>relativeToGround</altitudeMode>
<outerBoundaryIs><LinearRing><coordinates>-73.0,44.0 -71.0,44.0 -71.0,47.0
-73.0,47.0 -73.0,44.0 </coordinates>
</LinearRing></outerBoundaryIs></Polygon>"^^<http://www.opengis.net/ont/
geosparql#kmlLiteral>
```

`ogc:kmlLiteral` encodings consist of a valid KML geometry serialization. `ogc:kmlLiterals` are always interpreted using WGS84 geodetic longitude-latitude spatial reference system.

Several geometry types can be represented as geometry literal values, including point, linestring, polygon, polyhedral surface, triangle, TIN, multipoint, multi-linestring, multipolygon, and geometry collection. Up to 500,000 vertices per geometry are supported for two-dimensional geometries.

Example 1-88 Spatial Data Encoded Using ogc:wktLiteral Values

The following example shows some RDF spatial data (in N-triple format) encoded using `ogc:wktLiteral` values. In this example, the first two geometries (in lot1) use the default

WGS84 coordinate system (SRID 4326), but the other two geometries (in lot2) specify SRID 4269.

```
# spatial data for lot1 using the default WGS84 Longitude-Latitude spatial
reference system
<urn:lot1> <urn:hasExactGeometry> "Polygon((-83.6 34.1, -83.6 34.5, -83.2
34.5, -83.2 34.1, -83.6 34.1))"^^<http://www.opengis.net/ont/
geosparql#wktLiteral> .
<urn:lot1> <urn:hasPointGeometry> "Point(-83.4 34.3)"^^<http://
www.opengis.net/ont/geosparql#wktLiteral> .
# spatial data for lot2 using the NAD83 Longitude-Latitude spatial reference
system
<urn:lot2> <urn:hasExactGeometry> "<http://www.opengis.net/def/crs/
EPSG/0/4269> Polygon((-83.6  34.1, -83.6 34.3, -83.4 34.3, -83.4 34.1, -83.6
34.1))"^^<http://www.opengis.net/ont/geosparql#wktLiteral> .
<urn:lot2> <urn:hasPointGeometry> "<http://www.opengis.net/def/crs/
EPSG/0/4269> Point(-83.5 34.2)"^^<http://www.opengis.net/ont/
geosparql#wktLiteral> .
```

For more information, see the chapter about coordinate systems (spatial reference systems) in *Oracle Spatial Developer's Guide*. See also the material about the WKT geometry representation in the Open Geospatial Consortium (OGC) Simple Features document, available at: `http://www.opengeospatial.org/standards/sfa`

1.7.11.3 Validating Geometries

Before manipulating spatial data, you should check that there are no invalid geometry literals stored in your RDF graph. The procedure SEM_APIS.VALIDATE_GEOMETRIES allows verifying geometries in an RDF graph. The geometries are validated using an input SRID and tolerance value. (SRID and tolerance are explained in Indexing Spatial Data.)

If there are invalid geometries, a table with name {graph_name}_IVG$, is created in the user schema, where {graph_name} is the name of the RDF graph specified. Such table contains, for each invalid geometry literal, the value_id of the geometry literal in the RDF_VALUE$ table, the error message explaining the reason the geometry is not valid and a corrected geometry literal if the geometry can be rectified. For more information about geometry validation, see the reference information for the Oracle Spatial subprograms SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT and SDO_GEOM.VALIDATE_LAYER_WITH_CONTEXT.

Example 1-89 Validating Geometries in an RDF Graph

The following example validates an RDF graph `m`, using `SRID=8307` and `tolerance=0.1`.

```
-- Validate
EXECUTE sem_apis.validate_geometries(RDF graph_name=>'m',SRID=>8307,tolerance=>0.1,
network_owner=>'RDFUSER', network_name=>'NET1');-- Check for invalid geometries
SELECT original_vid, error_msg, corrected_wkt_literal FROM M_IVG$;
```

1.7.11.4 Indexing Spatial Data

Before you can use any of the SPARQL extension functions (introduced in Querying Spatial Data) to query spatial data, you must create a spatial index on the RDF network by calling the SEM_APIS.ADD_DATATYPE_INDEX procedure.

When you create the spatial index, you must specify the following information:

- SRID - The ID for the spatial reference system in which to create the spatial index. Any valid spatial reference system ID from Oracle Spatial and Graph can be used as an SRID value.

> **Note:**
>
> If you plan to use geospatial RDF data in web-based mapping applications like Oracle Spatial Studio, it is recommended to pre-transform your data to WGS84 longitude-latitude (SRID 4326 or 8307) and also use SRID 4326 or 8307 for your spatial index. This will improve performance by avoiding repeated coordinate transformations to WGS84 longitude-latitude for display on a map.

- TOLERANCE – The tolerance value for the spatial index. Tolerance is a positive number indicating how close together two points must be to be considered the same point. The units for this value are determined by the default units for the SRID used (for example, meters for WGS84 Long-Lat). Tolerance is explained in detail in *Oracle Spatial Developer's Guide*.
- DIMENSIONS - A text string encoding dimension information for the spatial index. Each dimension is represented by a sequence of three comma-separated values: name, minimum value, and maximum value. Each dimension is enclosed in parentheses, and the set of dimensions is enclosed by an outer parenthesis.

Example 1-90 Adding a Spatial Data Type Index on RDF Data

Example 1-90 adds a spatial data type index on the RDF network, specifying the WGS84 Longitude-Latitude spatial reference system, a tolerance value of 0.1, and the recommended dimensions for the indexing of spatial data that uses this coordinate system. The TOLERANCE, SRID, and DIMENSIONS keywords are case sensitive, and creating a data type index for any supported geometry literal type (`<http://xmlns.oracle.com/rdf/geo/WKTLiteral>`, `<http://www.opengis.net/ont/geosparql#wktLiteral>`, `<http://www.opengis.net/ont/geosparql#gmlLiteral>`, `<http://www.opengis.net/ont/geosparql#geoJSONLiteral>`, or `<http://www.opengis.net/ont/geosparql#kmlLiteral>`) will create an index for all the supported geometry literal types. For example, if you create an index for `ogc:wktLiteral`, any `orageo:WKTLiteral`, `ogc:gmlLiteral`, `ogc:geoJSONLiteral`, and `ogc:kmlLiteral` geometry literals will also be indexed.

```
EXECUTE sem_apis.add_datatype_index('http://www.opengis.net/ont/geosparql#wktLiteral',
                    options=>'TOLERANCE=0.1 SRID=8307 DIMENSIONS=((LONGITUDE,-180,180)
(LATITUDE,-90,90))',
                    network_owner=>'RDFUSER', network_name=>'NET1');
```

No more than one spatial data type index is supported for an RDF network. Geometry literal values stored in the RDF network are automatically normalized to the spatial reference system used for the index, so a single spatial index can simultaneously support geometry literal values from different spatial reference systems. This coordinate transformation is done transparently for indexing and spatial computations. When geometry literal values are returned from a SEM_MATCH query, the original, untransformed geometry is returned.

For more information about spatial indexing, see the chapter about indexing and querying spatial data in *Oracle Spatial Developer's Guide*.

Example 1-91 Adding a Spatial Data Type Materialized Index on RDF Data

When you manipulate spatial data, conversions from geometry literals to geometry objects may be needed, but several conversions may lead to poor performance. To avoid this situation, all the stored geometry literals can be transformed into SDO_GEOMETRY objects and materialized at index creation time.

This can be achieved using the `MATERIALIZE=T` option when adding a spatial data type index. If the amount of geometry literals to be indexed is very large, using the option `INS_AS_SEL=T` may help to speed up the materialized index creation.

The following example shows the creation of a materialized spatial index.

```
EXECUTE sem_apis.add_datatype_index('http://www.opengis.net/ont/
geosparql#wktLiteral',
             options=>'TOLERANCE=0.1 SRID=8307
DIMENSIONS=((LONGITUDE,-180,180) (LATITUDE,-90,90)) MATERIALIZE=T ');
```

Example 1-92 Adding a 3D Spatial Data Type Index on RDF Data

Spatial indexes with three coordinates can be created in Oracle Spatial. To create a 3D index, you must specify SDO_INDX_DIMS=3 option in the options argument of the SEM_APIS.ADD_DATATYPE_INDEX procedure.

The following example shows creation and indexing of 3D data. Note that coordinates are specified in (X, Y, Z) order, and linear rings for outer polygon boundaries are given in counter-clockwise order.

Note: For information about support for geometry operations with 3D data, including any restrictions, see Three Dimensional Spatial Objects.

```
conn rdfuser/<password>;

create table geo3d_tab(tri sdo_rdf_triple_s);

exec sem_apis.create_sem_model('geo3d','geo3d_tab','tri');

-- 3D Polygon
insert into geo3d_tab(tri) values(sdo_rdf_triple_s('geo3d','<http://
example.org/ApplicationSchema#A>',
                                  '<http://example.org/
ApplicationSchema#hasExactGeometry>',
                                  '<http://example.org/
ApplicationSchema#AExactGeom>'));
insert into geo3d_tab(tri) values(sdo_rdf_triple_s('geo3d','<http://
example.org/ApplicationSchema#AExactGeom>',
                                  '<http://www.opengis.net/ont/
geosparql#asWKT>',
                                  '"<http://xmlns.oracle.com/rdf/geo/srid/
31468> Polygon ((4467504.578 5333958.396 513.9,
                                                                 4467508.939
5333956.379 513.9,
                                                                 4467509.736
5333958.101 513.9,
                                                                 4467505.374
5333960.118 513.9,
                                                                 4467504.578
5333958.396 513.9))"^^<http://www.opengis.net/ont/geosparql#wktLiteral>'));

-- 3D Point at same elevation as Polygon
insert into geo3d_tab(tri) values(sdo_rdf_triple_s('geo3d','<http://
example.org/ApplicationSchema#B>',
                                                 '<http://example.org/
```

```
ApplicationSchema#hasExactGeometry>',
                                                    '<http://example.org/
ApplicationSchema#BExactGeom>'));
insert into geo3d_tab(tri) values(sdo_rdf_triple_s('geo3d','<http://
example.org/ApplicationSchema#BExactGeom>',
                                                    '<http://
www.opengis.net/ont/geosparql#asWKT>',
                                                    '"<http://
xmlns.oracle.com/rdf/geo/srid/31468> Point (4467505.000 5333959.000
513.9)"^^<http://www.opengis.net/ont/geosparql#wktLiteral>'));

-- 3D Point at different elevation from Polygon
insert into geo3d_tab(tri) values(sdo_rdf_triple_s('geo3d','<http://
example.org/ApplicationSchema#C>',
                                                    '<http://example.org/
ApplicationSchema#hasExactGeometry>',
                                                    '<http://example.org/
ApplicationSchema#CExactGeom>'));
insert into geo3d_tab(tri) values(sdo_rdf_triple_s('geo3d','<http://
example.org/ApplicationSchema#CExactGeom>',
                                                    '<http://
www.opengis.net/ont/geosparql#asWKT>',
                                                    '"<http://
xmlns.oracle.com/rdf/geo/srid/31468> Point (4467505.000 5333959.000
13.9)"^^<http://www.opengis.net/ont/geosparql#wktLiteral>'));
commit;

-- Create 3D index
conn system/manager;
exec sem_apis.add_datatype_index('http://www.opengis.net/ont/
geosparql#wktLiteral' ,
                                  options=>'TOLERANCE=0.1 SRID=3148
                                  DIMENSIONS=((x,4386596.4101,4613610.5843)
(y,5237914.5325,6104496.9694) (z,0,10000))
                                  SDO_INDX_DIMS=3 ');

conn rdfuser/rdfuser;
-- Find geometries within 200 M of my:A
-- Returns only one point because of 3D index
SELECT aGeom, f, fGeom, aWKT, fWKT
FROM TABLE(SEM_MATCH(
 '{ my:A my:hasExactGeometry ?aGeom .
     ?aGeom ogc:asWKT ?aWKT .
     ?f my:hasExactGeometry ?fGeom .
     ?fGeom ogc:asWKT ?fWKT .
     FILTER (orageo:withinDistance(?aWKT, ?fWKT,200,"M") &&
             !sameTerm(?aGeom,?fGeom))
   }',
 SEM_Models('geo3d'),
 null,
 SEM_ALIASES(
  SEM_ALIAS('my','http://example.org/ApplicationSchema#')),
 null));
```

1.7.11.5 Querying Spatial Data

Several SPARQL extension functions are available for performing spatial queries in SEM_MATCH. For example, for spatial RDF data, you can find the area and perimeter (length) of a geometry, the distance between two geometries, and the centroid and the minimum bounding rectangle (MBR) of a geometry, and you can check various topological relationships between geometries.

SEM_MATCH Support for Spatial Queries contains reference and usage information about the available functions, including:

- GeoSPARQL functions
- Oracle-specific functions

1.7.11.6 Using Long Literals with GeoSPARQL Queries

Geometry literals can become very long, which make the use of CLOBs necessary to represent them when using a SQL interface. CLOB constants cannot be used directly in a SEM_MATCH query. However, a user-defined SPARQL function can be used to bind CLOB constants into SEM_MATCH queries. Note that long geometry literals can be used directly in SPARQL query strings when using Java or REST interfaces for SPARQL execution.

The following example uses a user-defined SPARQL function in combination with a temporary table to allow CLOB geometries in a SEM_MATCH query.

Example 1-93 Binding a CLOB Constant into a SPARQL Query

```
conn rdfuser/<password>;

-- Create temporary table
create global temporary table local_value$(
 VALUE_TYPE      VARCHAR2(10),
 VALUE_NAME     VARCHAR2(4000),
 LITERAL_TYPE     VARCHAR2(1000),
 LANGUAGE_TYPE     VARCHAR2(80),
 LONG_VALUE      CLOB)
on commit preserve rows;

-- Create user-defined function to transform a CLOB into an RDF term
CREATE OR REPLACE FUNCTION myGetClobTerm
RETURN SDO_RDF_TERM
AS
  term SDO_RDF_TERM;
BEGIN
  select sdo_rdf_term(
      value_type,
      value_name,
      literal_type,
      language_type,
      long_value)
  into term
  from local_value$
  where rownum < 2;

  RETURN term;
```

```
END;
/

-- Insert a row with CLOB geometry
insert into local_value$
(value_type,value_name,literal_type,language_type,long_value)
values ('LIT','','http://www.opengis.net/ont/
geosparql#wktLiteral','','Some_CLOB_WKT');

-- Use the CLOB constant in a SEM_MATCH query
SELECT cdist
FROM table(sem_match(
'{ ?cdist ogc:asWKT ?cgeom
   FILTER (
     orageo:withinDistance(?cgeom, oraextf:myGetClobTerm(), 200, "M")) }'
,sem_models('gov_all_vm')
,null, null, null, null, ' ALLOW_DUP=T ', null, null
,'RDFUSER', 'NET1'));
```

1.7.12 Flashback Query Support

You can perform SEM_MATCH queries that return past data using Flashback Query. A TIMESTAMP or a System Change Number (SCN) value is passed to SEM_MATCH through the AS_OF hint. The AS_OF hint can have one of the following forms:

- `AS_OF[TIMESTAMP,<TIMESTAMP_VALUE>]`, where <TIMESTAMP_VALUE> is a valid timestamp string with format 'YYYY/MM/DD HH24:MI:SS.FF'.
- `AS_OF[SCN,<SCN_VALUE>]`, where <SCN_VALUE> is a valid SCN.

The AS_OF hint is internally transformed to perform a Flashback Query (SELECT AS OF) against the queried table or view containing triples of the specified RDF graph. This allows you to query the graph as it existed in a prior time. For this feature to work, the invoker needs a flashback privilege on the queried metadata table or view (RDFM_*rdf-graph-name* view for native RDF graphs, SEMU_*rdf-collection--name* and SEMV_*rdf-collection-name* for RDF graph collections, and underlying relational tables for RDF view graphs). For example: `grant flashback on RDFUSER.NET1#RDFM_FAMILY to scott`

Restrictions on Using Flashback Query with RDF Data

Adding or removing a partition from a partitioned table disables Flashback Query for previous versions of the partitioned table. As a consequence, creating or dropping a native RDF graph or creating or dropping an inferred graph will disable Flashback Query for previous versions of all native RDF graphs in an RDF network. Therefore, be sure to control such operations when using Flashback Query in an RDF network.

Example 1-94   Flashback Query Using TIMESTAMP

The following example shows the use of the AS_OF clause defining a TIMESTAMP.

```
SELECT x, name
  FROM TABLE(SEM_MATCH(
    'PREFIX : <http://www.example.org/family/>
     SELECT *
     WHERE { ?x :name ?name }',
    SEM_Models('family'),
    null, null,
    null,null,' AS_OF=[TIMESTAMP,2016/05/02 13:06:03.979546]',
```

```
    null, null,
    'RDFUSER', 'NET1'));
```

Example 1-95 Flashback Query Using SCN

The following example shows the use of the AS_OF clause specifying an SCN.

```
SELECT x, name
  FROM TABLE(SEM_MATCH(
    'PREFIX : <http://www.example.org/family/>
     SELECT *
     WHERE { ?x :name ?name }',
    SEM_Models('family'),
    null, null,
    null,null,' AS_OF=[SCN,1429849]',
    null, null,
    'RDFUSER', 'NET1'));
```

1.7.13 Best Practices for Query Performance

This section describes some recommended practices for using the SEM_MATCH table function to query RDF data. It includes the following subsections:

- FILTER Constructs Involving xsd:dateTime, xsd:date, and xsd:time
- Indexes for FILTER Constructs Involving Typed Literals
- FILTER Constructs Involving Relational Expressions
- Optimizer Statistics and Dynamic Sampling
- Multi-Partition Queries
- Compression on Systems with OLTP Index Compression
- Unbounded Property Path Expressions
- Nested Loop Pushdown for Property Paths
- Grouping and Aggregation
- Use of Bind Variables to Reduce Compilation Time
- Non-Null Expression Hints
- Automatic JOIN Hints
- RDF Network Indexes
- Using RDF with Oracle Database In-Memory
- Using Language Tags in FILTER Expressions
- Type Casting for More Efficient FILTER Evaluation
- Spatial Indexing for GeoSPARQL Queries

1.7.13.1 FILTER Constructs Involving xsd:dateTime, xsd:date, and xsd:time

By default, SEM_MATCH complies with the XML Schema standard for comparison of xsd:date, xsd:time, and xsd:dateTime values. According to this standard, when comparing two calendar values c1 and c2 where c1 has an explicitly specified time zone and c2 does not have a specified time zone, c2 is converted into the interval [c2-14:00, c2+14:00]. If c2-14:00 <= c1 <= c2+14:00, then the comparison is undefined and will always evaluate to false. If c1 is outside this interval, then the comparison is defined.

However, the extra logic required to evaluate such comparisons (value with a time zone and value without a time zone) can significantly slow down queries with FILTER constructs that involve calendar values. For improved query performance, you can disable this extra logic by specifying `FAST_DATE_FILTER=T` in the `options` parameter of the SEM_MATCH table function. When `FAST_DATE_FILTER=T` is specified, all calendar values without time zones are assumed to be in Greenwich Mean Time (GMT).

Note that using `FAST_DATE_FILTER=T` does *not* affect query *correctness* when either (1) all calendar values in the data set have a time zone or (2) all calendar values in the data set do not have a time zone.

1.7.13.2 Indexes for FILTER Constructs Involving Typed Literals

The evaluation of SEM_MATCH queries involving the FILTER construct often uses order columns on the RDF_VALUE$ table. For example, the filter `(?x < "1929-11-16Z"^^xsd:date)` uses the ORDER_DATE column.

Indexes can be used to improve the performance of queries that contain a filter condition involving a typed literal. For example, an `xsd:date` index may speed up evaluation of the filter `(?x < "1929-11-16Z"^^xsd:date)`.

Convenient interfaces are provided for creating, altering, and dropping these indexes for order columns. For more information, see Using Data Type Indexes.

Note, however, that the existence of these indexes on the RDF_VALUE$ table can significantly slow down bulk load operations. In many cases it may be faster to drop the indexes, perform the bulk load, and then re-create the indexes, as opposed to doing the bulk load with the indexes in place.

1.7.13.3 FILTER Constructs Involving Relational Expressions

The following recommendations apply to FILTER constructs involving relational expressions:

- The `orardf:sameCanonTerm` extension function is the most efficient way to compare two RDF terms for equality because it allows an id-based comparison in all cases.
- When using standard SPARQL features, the `sameTerm` built-in function is more efficient than using `=` or `!=` when comparing two variables in a FILTER clause, so (for example) use `sameTerm(?a, ?b)` instead of `(?a = ?b)` and use `(!sameTerm(?a, ?b))` instead of `(?a != ?b)` whenever possible.
- When comparing values in FILTER expressions, you may get better performance by reducing the use of negation. For example, it is more efficient to evaluate `(?x <= "10"^^xsd:int)` than it is to evaluate the expression `(!(?x > "10"^^xsd:int))`.

1.7.13.4 Optimizer Statistics and Dynamic Sampling

Having sufficient statistics for the query optimizer is critical for good query performance. In general, you should ensure that you have gathered basic statistics for the RDF network using the SEM_PERF.GATHER_STATS procedure (described in SEM_PERF Package Subprograms).

Due to the inherent flexibility of the RDF graph, static information may not produce optimal execution plans for SEM_MATCH queries. Dynamic sampling can often produce much better query execution plans. Dynamic sampling levels can be set at the session or system level using the `optimizer_dynamic_sampling` parameter, and at the individual query level using the `dynamic_sampling(level)` SQL query hint. In general, it is good to experiment with dynamic sampling levels between 3 and 6. For information about estimating statistics with dynamic sampling, see *Oracle Database SQL Tuning Guide*.

Example 1-96 uses a SQL hint for a dynamic sampling level of 6.

Example 1-96   SQL Hint for Dynamic Sampling

```
SELECT /*+ DYNAMIC_SAMPLING(6) */ x, y
  FROM TABLE(SEM_MATCH(
    'PREFIX : <http://www.example.org/family/>
     SELECT *
     WHERE {
      ?x :grandParentOf ?y .
      ?x rdf:type :Male .
      ?x :birthDate ?bd }',
    SEM_Models('family'),
    SEM_Rulebases('RDFS','family_rb'),
    null, null, null, '', null, null,
    'RDFUSER', 'NET1'));
```

1.7.13.5 Multi-Partition Queries

The following recommendations apply to the use of multiple RDF graphs, RDF graphs plus inferred graphs, and RDF graph collections:

- If you execute SEM_MATCH queries against multiple RDF graphs or against RDF graphs plus inferred graphs, you can probably improve query performance if you create a RDF graph collection (see RDF Graph Collections) that contains all the RDF graphs and inferred graphs you are querying and then query this single RDF graph collection.
- Use the `ALLOW_DUP=T` query option. If you do not use this option, then an expensive (in terms of processing) duplicate-elimination step is required during query processing, in order to maintain set semantics for RDF data. However, if you use this option, the duplicate-elimination step is not performed, and this results in significant performance gains.

1.7.13.6 Compression on Systems with OLTP Index Compression

On systems where OLTP index compression is supported (such as Exadata). you can take advantage of the feature to improve the compression ratio for some of the B-tree indexes used by the RDF network.

For example, a DBA or the owner of a schema-private network can use the following command to change the compression scheme on the RDF_VAL_NAMETYLITLNG_IDX index from prefix compression to OLTP index compression:

```
SQL> alter index rdfuser.net1#RDF_VAL_NAMETYLITLNG_IDX rebuild compress for oltp high;
```

1.7.13.7 Unbounded Property Path Expressions

A depth-limited search should be used for + and * property path operators whenever possible. The depth-limited implementation for * and + is likely to significantly outperform the CONNECT BY-based implementation in large and/or highly connected graphs. A depth limit of 10 is used by default. For a given graph, depth limits larger than the graph's diameter are not useful. See Property Paths for more information on setting depth limits.

A backward chaining style inference using `rdfs:subClassOf+` for ontologies with very deep class hierarchies may be an exception to this rule. In such cases, unbounded CONNECT BY-based evaluations may perform better than depth-limited evaluations with very high depth limits (for example, 50).

1.7.13.8 Nested Loop Pushdown for Property Paths

If an unbounded CONNECT BY evaluation is performed for a property path, and if the subject of the property path triple pattern is a variable, a CONNECT BY WITHOUT FILTERING operation will most likely be used. If this subject variable is only bound to a small number of values during query execution, a nested loop strategy (see Nested Loop Pushdown with Overloaded Service) could be a good option to run the query. In this case, the property path can be pushed down into an overloaded SERVICE clause and the OVERLOADED_NL=T hint can be used.

For example, consider the following query where there is an unbounded property path search `{ ?s :hasManager+ ?x }`, but the triple `{ ?s :ename "ADAMS" }` only has a small number of possible values for `?s`.

```
select s, x
from table(sem_match(
'PREFIX : <http://scott-hr.org#>
 SELECT *
 WHERE {
   ?s :ename "ADAMS" .
   ?s :hasManager+ ?x .
 }',
sem_models('scott_hr_data'),
null,null,null,null,'  ALL_MAX_PP_DEPTH(0) ', null, null,
'RDFUSER', 'NET1'));
```

The query can be transformed to force the nested-loop strategy. Notice that the RDF graph specified in the SERVICE graph is the same as the RDF graph specified in the SEM_MATCH call.

```
select s, x
from table(sem_match(
'PREFIX : <http://scott-hr.org#>
 SELECT *
 WHERE {
   ?s :ename "ADAMS" .
   service oram:scott_hr_data { ?s :hasManager+ ?x . }
 }',
sem_models('scott_hr_data'),
null,null,null,null,'  ALL_MAX_PP_DEPTH(0) OVERLOADED_NL=T ', null, null,
'RDFUSER', 'NET1'));
```

With this nested-loop strategy, `{ ?s :hasManager_ ?x }` is evaluated once for each value of `?s`, and in each evaluation, a constant value is substituted for `?s`. This constant in the subject position allows a CONNECT BY WITH FILTERING operation, which usually provides a substantial performance improvement.

1.7.13.9 Grouping and Aggregation

`MIN`, `MAX` and `GROUP_CONCAT` aggregates require special logic to fully capture SPARQL semantics for input of non-uniform type (for example, `MAX(?x)`). For certain cases where a uniform input type can be determined at compile time (for example, `MAX(STR(?x))` – plain literal input), optimizations for built-in SQL aggregates can be used. Such optimizations generally give an order of magnitude increase in performance. The following cases are optimized:

- MIN/MAX(<plain literal>)
- MIN/MAX(<numeric>)
- MIN/MAX(<dateTime>)
- GROUP_CONCAT(<plain literal>)

Example 1-97 uses MIN/MAX(<numeric>) optimizations.

Example 1-97 Aggregate Optimizations

```
SELECT dept, minSal, maxSal
  FROM TABLE(SEM_MATCH(
    'SELECT ?dept (MIN(xsd:decimal(?sal)) AS ?minSal) (MAX(xsd:decimal(?sal)) AS ?maxSal)
     WHERE
       {?x :salary ?y .
        ?x :department ?dept }
     GROUP BY ?dept',
    SEM_Models('hr_data'),
    null, null, null, null, '', null, null,
    'RDFUSER', 'NET1'));
```

1.7.13.10 Use of Bind Variables to Reduce Compilation Time

For some queries, query compilation can be more expensive than query execution, which can limit throughput on workloads of small queries. If the queries in your workload differ only in the constants used, then session context-based bind variables can be used to skip the compilation step for SEM_MATCH queries. See also Using Bind Variables with SEM_APIS.SPARQL_TO_SQL for a description of how to use JDBC bind variables and PL/SQL bind variables with SPARQL queries.

The following example shows how to use a session context in combination with a user-defined SPARQL function to compile a SEM_MATCH query once and then run it with different constants. The basic idea is to create a user-defined function that reads an RDF term value from the session context and returns it. A SEM_MATCH query with this function will read the RDF term value at run time; so when the session context variable changes, the same exact SEM_MATCH query will see a different value.

```
conn / as sysdba;
grant create any context to testuser;

conn testuser/testuser;

create or replace package MY_CTXT_PKG as
  procedure set_attribute(name varchar2, value varchar2);
  function get_attribute(name varchar2) return varchar2;
end MY_CTXT_PKG;
/

create or replace package body MY_CTXT_PKG as
  procedure set_attribute(
    name varchar2,
    value varchar2
  ) as
```

```
  begin
    dbms_session.set_context(namespace => 'MY_CTXT',
                             attribute => name,
                             value     => value );
  end;

  function get_attribute(
    name varchar2
  ) return varchar2 as
  begin
    return sys_context('MY_CTXT', name);
  end;
end MY_CTXT_PKG;
/

create or replace function myCtxFunc(
  params in SDO_RDF_TERM_LIST
) return SDO_RDF_TERM
as
  name varchar2(4000);
  arg  SDO_RDF_TERM;
begin
  arg := params(1);
  name := arg.value_name;
  return SDO_RDF_TERM(my_ctxt_pkg.get_attribute(name));
end;
/

CREATE OR REPLACE CONTEXT MY_CTXT using TESTUSER.MY_CTXT_PKG;

-- Set a value
exec MY_CTXT_PKG.set_attribute('value','<http://www.example.org/family/
Martha>');

-- Query using the function
-- Note the use of HINT0={ NON_NULL } to allow the most efficient join
SELECT s, p, o
  FROM TABLE(SEM_MATCH(
    'SELECT ?s ?p ?o
     WHERE {
       BIND (oraextf:myCtxFunc("value") # HINT0={ NON_NULL }
             AS ?s)
       ?s ?p ?o }',
    SEM_Models('family'),
    null,
    null,
    null, null, ' ', null, null,
    'RDFUSER', 'NET1'));

-- Set another value
exec MY_CTXT_PKG.set_attribute('value','<http://www.example.org/family/
Sammy>');

-- Now the same query runs for Sammy without recompiling
SELECT s, p, o
  FROM TABLE(SEM_MATCH(
```

```
    'SELECT ?s ?p ?o
     WHERE {
       BIND (oraextf:myCtxFunc("value") # HINT0={ NON_NULL }
             AS ?s)
       ?s ?p ?o }',
    SEM_Models('family'),
    null,
    null,
    null, null, ' ', null, null,
    'RDFUSER', 'NET1'));
```

1.7.13.11 Non-Null Expression Hints

When performing a join of several graph patterns with common variables that can be unbound, a more complex join condition is needed to handle null values to avoid performance degradation. Unbound values can be introduced through SELECT expressions, binds, OPTIONAL clauses, and unions. In many cases, SELECT expressions are not expected to produce NULL values. In such cases, query performance can be substantially improved through use of an inline HINT0={ NON_NULL } hint to mark a specific SELECT expression as definitely non-null or through use of a DISABLE_NULL_EXPR_JOIN query option to signify that all SELECT expressions produce only non-null values.

The following example includes the global DISABLE_NULL_EXPR_JOIN hint to signify that variable `?fulltitle` is always bound on both sides of the join. (See also Inline Query Optimizer Hints.)

```
SELECT s, t
  FROM TABLE(SEM_MATCH(
    'PREFIX : <http://www.example.org/family/>
     SELECT * WHERE {
       { SELECT ?s (CONCAT(?title, ". ", ?fullname) AS ?fulltitle)
         WHERE { ?s :fullname ?fullname .
                 ?s :title ?title }
       }
       { SELECT ?t (CONCAT(?title, ". ", ?fname, " ", ?lname) AS ?fulltitle)
         WHERE {
         ?t :fname ?fname .
         ?t :lname ?lname .
         ?t :title ?title }
       }
     }',
    SEM_Models('family'),
    SEM_Rulebases('RDFS','family_rb'),
    null,
    null,
    null,
    ' DISABLE_NULL_EXPR_JOIN ', null, null,
    'RDFUSER', 'NET1'));
```

1.7.13.12 Automatic JOIN Hints

SEM_MATCH queries that are very unselective usually execute faster if the SQL engine uses HASH joins to evaluate joins between triple patterns. The SPARQL-to-SQL query translator used by SEM_MATCH will attempt to auto detect such queries and automatically add appropriate `USE_HASH` hints if the string `AUTO_HINTS=T` appears in the options argument string.

The following SEM_MATCH query uses `AUTO_HINTS=T` to automatically generate `USE_HASH` hints.

```
SELECT f, l, n, e
  FROM table(sem_match(
        'PREFIX : <http://www.example.com#>
         SELECT ?f ?l ?n ?e
          WHERE { ?s :fname ?f . ?s :lname ?l . ?s :nickName ?n . ?s :email ?e }',
                sem_models('m1'),
                null,null,null,null,
                ' AUTO_HINTS=T ')
          );
```

1.7.13.13 RDF Network Indexes

RDF Network Indexes (described in Using RDF Network Indexes) are nonunique B-tree indexes on the RDF_LINK$ table. Network owners and DBAs can manage these indexes with various SEM_APIS procedures. Columns to index in RDF_LINK$ are identified by an index code, which is a sequence of the following letters (without repetition): `P`, `C`, `S`, `G`, `M`, `H`. These letters used in the `index_code` correspond to the following columns in RDF_LINK$: P_VALUE_ID (predicate), CANON_END_NODE_ID (object), START_NODE_ID (subject), G_ID (graph), MODEL_ID, and H - a function-based index on (MODEL_ID, GID).

It is important to have the proper set of RDF Network Indexes for your query workload. In versions 19c and earlier, the default index setup is `PCSGM`, `PSCGM`. In versions 21c and later the default index setup is `PCSGM`, `SPCGM`, `CM`, `H`.

The following are a few general recommendations for RDF Network Indexes:

- Most SPARQL queries have triple patterns with bound predicates, so it is a good idea to have `P`, `PC`, and `PS` combinations covered as leading columns in your overall index set. Such a combination is captured by the default index setup (`PCSGM`, `PSCGM` in 19c, and `PCSGM`, `SPCGM` in 21c).
- If you have queries with unbound predicates (for example, `{ ?s :ssn 1234 . ?s ?p ?o }`), then a network index with a leading column other than `P` may be needed. An `SPCGM` index would be more suitable for this example because of the join on subject variable `?s`.
- If you are running DESCRIBE queries or DESCRIBE-style patterns such as `{ { <urn:abc> ?p1 ?o1 } UNION { ?s2 ?p2 <urn:abc> } }`, then a network index with a leading `C` column (for example, `CM`) in addition to an index with a leading `S` column may be needed.
- If you have named graph queries with selective FROM, FROM NAMED, or GRAPH clauses, then a network index with a leading `G` column may be needed (for example, `GPCSM`).
- An `H` index is needed for efficient SPARQL Update GRAPH operations (for example, DROP GRAPH) on schema-private networks.
- A `PSCGM` index is usually smaller than an `SPCGM` index due to better prefix compression, so if your workload does not include queries with unbound predicates, replacing an `SPCGM` index with a `PSCGM` index may give better performance.

1.7.13.14 Using RDF with Oracle Database In-Memory

RDF data stored in the RDF_LINK$ and RDF_VALUE$ tables can be loaded into memory using Oracle Database In-Memory. See RDF Support for Oracle Database In-Memory for details on how to load RDF data into memory using SEM_APIS procedures.

In general, for the best and most consistent performance with Oracle Database In-Memory, it is recommended to make indexes on the RDF_LINK$ (RDF network indexes) and RDF_VALUE$ tables invisible, with the exception of <NETWORK_NAME>#C_PK_VID and <NETWORK_NAME>#RDF_VAL_NAMETYLITLNG_IDX indexes on RDF_VALUE$. These index settings can be achieved with the following SQL commands (assuming a RDF network named NET1 owned by RDFUSER).

```
exec sem_apis.alter_rdf_indexes('VISIBILITY','N', network_owner=>'RDFUSER',
network_name=>'NET1');

alter index NET1#C_PK_VID visible;

alter index NET1#RDF_VAL_NAMETYLITLNG_IDX visible;
```

Note that the performance of very selective queries may suffer with RDF_LINK$ indexes invisible, so you may need to experiment with index visibility depending on your query workload.

In addition to these index settings, it is recommended to use parallel query execution with Oracle Database In-Memory, as the speedup from parallelization can be significant in many cases.

For larger datasets (100 M triples or more), it is also recommended to use a hash-subpartitioned RDF network with Oracle Database In-Memory. Hash subpartitioning is described in RDF Networks.

1.7.13.15 Using Language Tags in FILTER Expressions

When filtering query results based on language tags, it is more efficient to use LANG instead of LANGMATCHES whenever possible. For example, the simple filter `langMatches(lang(?x), "en")` could be replaced with `lang(?x) = "en"` for a more efficient evaluation. Language tags in stored RDF literals are canonicalized to lower case, so a lower case language tag constant should be used in such filters.

1.7.13.16 Type Casting for More Efficient FILTER Evaluation

SPARQL FILTERs that compare two variables using operators other than equality, for example `?x < ?y`, can have poor performance in some cases because of weak typing in SPARQL. Because datatypes for `?x` and `?y` cannot be determined at query compilation time, complex logic for comparisons of multiple datatypes must be used at run time.

If you know the datatypes of the values to which `?x` and `?y` will be bound, then it is best to cast `?x` and `?y` to those datatypes in your FILTER expression, so that the types will be known at query compilation time. For example, the following query casts salary values to `xsd:decimal` in the FILTER clause for a more efficient single-datatype comparison.

```
SELECT ?y
WHERE {
```

```
  :emp1 :salary ?s1 .
  ?y :salary ?s2 .
  FILTER (xsd:decimal(?s2) < xsd:decimal(?s1))
}
```

1.7.13.17 Spatial Indexing for GeoSPARQL Queries

Options used during spatial index creation can have significant effects on the performance of GeoSPARQL queries.

The two most important options are:

- **Type of index:** function-based or materialized
- **Spatial reference system:** SRID used for the index

SEM_APIS.ADD_DATATYPE_INDEX creates a function-based spatial index by default. A function-based index is adequate for simple point geometries, but you should use a materialized spatial index if your dataset contains polygon or line geometries. You can create a materialized spatial index by specifying `MATERIALIZE=T` in the options argument of SEM_APIS.ADD_DATATYPE_INDEX.

The SRID used for a spatial index is also important for performance. Oracle's GeoSPARQL implementation is very flexible in that it allows you to load geometry literals that have been encoded in different spatial reference systems. These geometries must be canonicalized to a single SRID for indexing and query evaluation. You can specify this canonical SRID at index creation time. For best performance, you must choose the SRID that is most common among your geometry literals to minimize required coordinate transformations.

See Indexing Spatial Data for more information on spatial index creation.

1.7.14 Special Considerations When Using SEM_MATCH

The following considerations apply to SPARQL queries executed using SEM_MATCH:

- Value assignment
  - A compile-time error is raised when undefined variables are referenced in the source of a value assignment.
- Grouping and aggregation
  - Non-grouping variables (query variables not used for grouping and therefore not valid for projection) cannot be reused as a target for value assignment.
  - Non-numeric values are ignored by the AVG and SUM aggregates.
  - By default, SEM_MATCH returns no rows for an aggregate query with a graph pattern that fails to match. The W3C specification requires a single, null row for this case. W3C-compliant behavior can be obtained with the `STRICT_AGG_CARD=T` query option for a small performance penalty.
- ORDER BY
  - When using SPARQL ORDER BY in SEM_MATCH, the containing SQL query should be ordered by SEM$ROWNUM to ensure that the desired ordering is maintained through any enclosing SQL blocks.
- Numeric computations
  - The native Oracle NUMBER type is used internally for all arithmetic operations, and the results of all arithmetic operations are serialized as `xsd:decimal`. Note that the

native Oracle NUMBER type is more precise than both BINARY_FLOAT and BINARY_DOUBLE. See *Oracle Database SQL Language Reference* for more information on the NUMBER built-in data type.

  – Division by zero causes a runtime error instead of producing an unbound value.

- Negation
  – EXISTS and NOT EXISTS filters that reference *potentially unbound variables* are not supported in the following contexts:
    * Non-aliased expressions in GROUP BY
    * Input to aggregates
    * Expressions in ORDER BY
    * FILTER expressions within OPTIONAL graph patterns that also reference variables that do not appear inside of the OPTIONAL graph pattern

    The first three cases can be realized by first assigning the result of the EXISTS or NOT EXISTS filter to a variable using a BIND clause or SELECT expression.

    These restrictions do *not* apply to EXISTS and NOT EXISTS filters that only reference definitely bound variables.
- Blank nodes
  – Blank nodes are not supported within graph patterns.
  – The `BNODE(literal)` function returns the same blank node value every time it is called with the same literal argument.
- Property paths
  – Unbounded operators + and * use a 10-hop depth limit by default for performance reasons. This behavior can be changed to a truly unbounded search by setting a depth limit of 0. See Property Paths for details.
- Long literals (CLOBs)
  – SPARQL functions and aggregates do not support long literals by default.
  – Specifying the `CLOB_EXP_SUPPORT=T` query option enables long literal support for the following SPARQL functions: IF, COALESCE, STRLANG, STRDT, SUBSTR, STRBEFORE, STRAFTER, CONTAINS, STRLEN, STRSTARTS, STRENDS.
  – Specifying the `CLOB_AGG_SUPPORT=T` query option enables long literal support for the following aggregates: MIN, MAX, SAMPLE, GROUP_CONCAT.
- Canonicalization of RDF literals
  – By default, RDF literals returned from SPARQL functions and constant RDF literals used in value assignment statements (BIND, SELECT expressions, GROUP BY expressions) are canonicalized. This behavior is consistent with the SPARQL 1.1 D-Entailment Regime.
  – Canonicalization can be disabled with the `PROJ_EXACT_VALUES=T` query option.

1.8 Speeding up Query Execution with Result Tables

Result tables are auxiliary tables that store the results for generic patterns of SPARQL queries executed against an RDF graph or RDF graph collection.

> **Note:**
>
> **Result tables** were called as **Subject-Property-Matrix (SPM) tables** in the previous book versions (prior to Oracle Database Release 23ai). See Changes in Terminology and Subprograms for more information.

Generic pattern queries include star-pattern, chain-pattern, and single-triple-pattern queries.

Improvement of performance with result tables is derived from the use of:

- Pre-materialized joins in these tables to reduce joins at query processing time
- Compact representation of triples for individual properties in separate tables for faster access
- More accurate RDF data statistics obtained from these tables to arrive at better query execution plans

The following sections provide in-depth information on result tables.

- Types of Result Tables
  There are three types of result tables that can be defined on an RDF graph or an RDF graph collection.
- Creating and Managing Result Tables
  The following sections explain the steps for creating and managing result tables.
- SPARQL Query Options for Result Tables
  SPARQL queries will automatically use result tables if they are present.
- Special Considerations when Using Result Tables
  This section describes a few limitations to be considered when using result tables.

1.8.1 Types of Result Tables

There are three types of result tables that can be defined on an RDF graph or an RDF graph collection.

The different result tables are as follows:

- **Star-Pattern Tables:** These tables hold the results for star-pattern queries (with restriction that each property must be single-valued) such as:
  `?x :fname ?fnm . ?x miname ?m . ?x :lname ?lnm .`
- **Triple-Pattern Tables:** These tables hold the results for single triple-pattern queries such as:
  `?x :hasHobby ?y .`

  This is same as an RDF triple, but for a specific property.
- **Chain-Pattern Tables:** These tables hold the results for chain-pattern queries such as:
  `?child :hasParent ?parent . ?parent :hasBrother ?uncle .`

  A chain is stored only if all the links exist.

> **Note:**
>
> **Star-Pattern, Triple-Pattern, and Chain-Pattern tables** were called as **Single-Valued Property (SVP), Multi-Valued Property (MVP), and Property Chain (PCN) tables** respectively in the previous book versions (prior to Oracle Database Release 23ai). See Changes in Terminology and Subprograms for more information.

Consider an RDF graph containing the following sample data.

```
:john :fname "John" ; :lname "Brown" ; :height 72 ; :email "john@email-
example.com", "johnnyB@email-example.com" .
:mary :fname "Mary" ; :lname "Smith" ; :height 68 ; :email "Mary.Smith@email-
example.com" .
:bob :fname "Robert" ; :lname "Brown" ; :height
70 ; :fatherOf :john, :mary ; :email "bobBrown@email-example.com" .
:alice :fname "Alice" ; :lname "Brown" ; :height 68 ; :motherOf :john, :mary .
:henry :fatherOf :bob .
:kathy :motherOf :bob .
```

Note that for simplicity, *Id(rdfterm)* will be used instead of the actual numeric identifier (available in the `RDF_VALUE$` table) for each *rdfterm*. A complete example with additional data is included in Example 1-107.

- Star-Pattern Tables
  Each row in a star-pattern table holds values for one or more single-valued RDF properties for a resource in an RDF graph.
- Triple-Pattern Tables
  Each row in a triple-pattern table, created for a given property, holds a value for the property.
- Chain-Pattern Tables
  Each row in a chain-pattern table holds a fixed-length *path* in the RDF graph.

1.8.1.1 Star-Pattern Tables

Each row in a star-pattern table holds values for one or more single-valued RDF properties for a resource in an RDF graph.

In the best case, a star-pattern table defined for `n` properties may be used during query processing to replace an `n`-way join of the `RDF_LINK$` table with simple table lookups.

A property `p` is single-valued in an RDF graph if each resource in the graph has at most one value for `p` regardless of named graphs. In the sample RDF dataset (described in Types of Result Tables), the properties `:first_name`, `:last_name`, and `:height` are single-valued, but the property `:email` is multi-valued.

To speed up execution of a query pattern such as *`{ ?s :first_name ?fname ; :last_name ?lname ; :height ?height }`*, involving use of single-valued properties only, a star-pattern table may be created on the RDF graph to include the preceding three single-valued properties by using the string *`':first_name :last_name :height'`* as the value for the *`key_string`* parameter in a call to the SEM_APIS.BUILD_RESULT_TAB subprogram.

Table 1-19 describes the structure and content for such a star-pattern table corresponding to the preceding sample data. Also, note that:

- The table shows only a subset of the actual set of columns. Specifically, not shown are the columns with name like **`G<Id(property)>`** that are used to store the named graph component of the corresponding RDF statements.
- The table describes values in the columns as `Id(rdfterm)` instead of the actual numeric identifiers that get stored.

Table 1-19 Example Star-Pattern Table Structure

START_NODE_ID	...	P<Id(:first_name)>	...	P<Id(:last_name)>	...	P<Id(:height)>
Id(:john)	...	Id("John")	...	Id("Brown")	...	Id(72)
Id(:mary)	...	Id("Mary")	...	Id("Smith")	...	Id(68)
Id(:bob)	...	Id("Robert")	...	Id("Brown")	...	Id(70)
Id(:alice)	...	Id("Alice")	...	Id("Brown")	...	Id(68)

The availability of this star-pattern table allows the preceding query pattern to be processed simply by accessing the rows in the star-pattern table and avoids the three-way self-join of the RDF_LINK$ table that would otherwise be necessary.

It is also possible to include *reversed-properties* that are single-valued. In the sample RDF data (described in Types of Result Tables), the property `:fatherOf` is not single-valued, but its reversed version which is denoted as `^:fatherOf` (intuitively equivalent to a `:hasFather` property), is indeed single-valued. To speed up execution of a query pattern such as *`{ ?s :fname ?fname; :lname ?lname; :height ?height; ^:fatherOf ?father }`*, an extended version of the preceding star-pattern table may be created, by using *`':fname :lname :height ^:fatherOf'`* as the *`key_string`* value.

Table 1-20 describes the structure and content of this extended version of the star-pattern table that includes a reversed property. The use of the letter `R`, instead of `P`, as the first character in the column name, `R<Id(:fatherOf)>`, indicates that this is a reversed property. As mentioned earlier, availability of this star-pattern table allows avoiding a (four-way) self-join of the `RDF_LINK$` table.

Table 1-20 Extended Star-Pattern Table Including a Reversed Property

START_NODE_ID	...	P<Id(:first_name)>	...	P<Id(:last_name)>	...	P<Id(:height)>	...	R<Id(:fatherOf)>
Id(:john)	...	Id("John")	...	Id("Brown")	...	Id(72)	...	:bob
Id(:mary)	...	Id("Mary")	...	Id("Smith")	...	Id(68)	...	:bob
Id(:bob)	...	Id("Robert")	...	Id("Brown")	...	Id(70)	...	:henry

Example 1-98 Creating a Star-Pattern Table

The following code creates an extended star-pattern table on an RDF graph named `M1`:

```
BEGIN
  SEM_APIS.BUILD_RESULT_TAB(
     query_pattern_type => SEM_APIS.SPM_TYPE_SVP
   , result_tab_name     => 'FLHF'
   , rdf_graph_name      => 'M1'
   , key_string          => ' :fname :lname :height ^:fatherOf '
   , prefixes            => ' PREFIX : <http://www.example.com#> '
   , network_owner       => 'RDFUSER'
   , network_name        => 'NET1'
```

```
  );
END;
/
```

The name, structure, and default indexes for a star-pattern table may be described as follows:

- The name of a star-pattern table is created based on the following template: `<NETWORK_NAME>#RDF_XT$`**`SVP`**`_<MODEL_NAME>+__<SPM_NAME>`
- The `NUMBER` column, `START_NODE_ID`, stores the subject id or, if reversed, the object id, of the matching triple for the first property in the list of properties in the star-pattern table.
- For each property covered in a star-pattern table, the following columns are created for storing the numeric identifiers for the lexical values in a triple: :
  - `NUMBER` column (`G<Id(property)>`) for storing the named graph id.
  - `NUMBER` column (`P<Id(property)>` for storing the object id or if reversed `R<Id(property)>`), the subject id.
  - (Optional) additional columns for internal use.
- The `START_NODE_ID` column is defined as the primary key of the star-pattern table and a unique index named using the template: `<NETWORK_NAME>#RDF_XX$SVP_<MODEL_NAME>_UQ__<SPM_NAME>`, is created on this column when the star-pattern table is created.

1.8.1.2 Triple-Pattern Tables

Each row in a triple-pattern table, created for a given property, holds a value for the property.

A triple-pattern table stores the values for a given property in a separate table and in a compact fashion, thus allowing faster access and better statistics. Unlike a star-pattern table, the (single) property included in a triple-pattern table does not have to be, but could be, single-valued.

A property `p` is multi-valued in an RDF graph if there exist two or more triples (regardless of named graphs), `(s p o1)` and `(s p o2)` with `o1` not equal to `o2`. That is, `s` has more than one distinct object values for the property `p`.

In the sample RDF dataset (described in Types of Result Tables), the properties `:email`, `:fatherOf`, and `:motherOf` are multi-valued.

Table 1-21 shows the structure and content of a triple-pattern table for the `:motherOf` property for the preceding sample data. The two columns shown here store the numeric identifiers for lexical values for variables `?mom` and `?c`, respectively, in a pattern `{ ?mom :motherOf ?c }`. The triple-pattern table contains another column, `G<id<:motherOf>)`, not shown here, to store the numeric identifier of the named graph in case the matching RDF statement is a quad.

Table 1-21 Example Triple-Pattern Table Structure

START_NODE_ID	...	P<Id(:motherOf)>
`Id(:alice)`		`Id(:john)`
`Id(:alice)`		`Id(:mary)`
`Id(:kathy)`		`Id(:bob)`

Example 1-99 Creating a Triple-Pattern Table

To create the preceding triple-pattern table on an RDF graph named M1, you can use the following SQL command.

```
BEGIN
  SEM_APIS.BUILD_RESULT_TAB(
     query_pattern_type => SEM_APIS.SPM_TYPE_MVP
   , result_tab_name    => null /* must be NULL (the name is auto-generated
based on id(property) */
   , rdf_graph_name     => 'M1'
   , key_string         => ' :motherOf ' /* must have exactly one property */
   , prefixes           => ' PREFIX : <http://www.example.com#> '
   , network_owner      => 'RDFUSER'
   , network_name       => 'NET1'
  );
END;
/
```

The name, structure, and default indexes for a triple-pattern table may be described as follows:

- The naming convention for a triple-pattern table is created based on the following template: `<NETWORK_NAME>#RDF_XT$`**`MVP`**`_<MODEL_NAME>+__P<id(property)>`
- The `NUMBER` column, `START_NODE_ID`, stores the subject id of the matching triples that use the target property as the predicate.
- For the property covered in a triple-pattern table, the following columns are created for storing the numeric identifiers for the lexical values in a triple:
  - `NUMBER` column `G<Id(property)>` for storing the named graph id
  - `NUMBER` column `P<Id(property)>` for storing the object id
  - Optional additional columns for internal use
- A nonunique index is created on the `START_NODE_ID` column using the following naming convention: `<NETWORK_NAME>#RDF_XX$MVP_<MODEL_NAME>_P<id(property)>` .

1.8.1.3 Chain-Pattern Tables

Each row in a chain-pattern table holds a fixed-length *path* in the RDF graph.

A path is a sequence of two or more triples where, except for the last triple in the sequence, object of a triple is the same as the subject of the next triple. A chain-pattern table that stores paths of length `n` can be used during query processing to replace an `n`-way join of type *current_triple.object* = *next_triple.subject*, of the `RDF_LINK$` table with simple table lookups.

For example, to speed up the execution of the following query pattern - `{ ?gma :motherOf ?f . ?f :fatherOf ?c }`, you can create a chain-pattern table using the following sequence of properties, specified as the `key_string`: `' :motherOf :fatherOf '`.

Table 1-22 shows the structure and content of the chain-pattern table for the preceding sample data. The three columns here store the numeric identifiers for lexical values for variables `?gma, ?f, and ?c`, respectively, for the two paths that satisfy the property chain: `(:kathy) -[:motherOf]-> (:bob) -[:fatherOf]-> (:john)` and `(:alice) -[:motherOf]-> (:bob) -[:fatherOf]-> (:mary)`.

Table 1-22 Example Chain-Pattern Table Structure

START_NODE_ID	...	P<Id(:motherOf)>	...	P<Id(:fatherOf)>
Id(:kathy)	...	Id(:bob)	...	Id(:john)
Id(:kathy)	...	Id(:bob)	...	Id(:mary)

A property chain can include *multiple occurrences* of the same property. Consider the following query pattern to connect a grandfather to the children:

```
{ ?gfa :fatherOf ?f . ?f :fatherOf ?c }
```

You can create a chain-pattern table using the following sequence of properties, specified as the key_string- ‘ :fatherOf :fatherOf ’. The following table describes the structure and content for such a chain-pattern table. The column name with ‘#2’ as suffix corresponds to the second occurrence of the :fatherOf property in the specified chain. It stores two paths that satisfy the property chain - (:henry) -[:fatherOf]-> (:bob) -[:fatherOf]-> (:john) and (:henry) -[:fatherOf]-> (:bob) -[:fatherOf]-> (:mary).

Table 1-23 Multiple Occurrences of a Single Property in a Chain-Pattern Table

START_NODE_ID	...	P<Id(:fatherOf)>	...	P<Id(:fatherOf)>#2
Id(:henry)	...	Id(:bob)	...	Id(:john)
Id(:henry)	...	Id(:bob)	...	Id(:mary)

A property chain may involve reversed properties as well. For example, consider the following query pattern { ?mom :motherOf ?c . ?c ^:fatherOf ?dad } to connect the siblings. You can create a Chain-Pattern table with the following key_string- ‘:motherOf ^:fatherOf ’.

Table 1-24 shows the structure and content of this chain-pattern table. Note that the letter ‘R’ in the rightmost column name R<id(:fatherOf)> indicates that the column corresponds to the reversed property. The availability of this chain-pattern table allows the preceding query pattern to be processed simply by accessing the rows in the chain-pattern table and avoids the two-way join of the RDF_LINK$ table that would otherwise be necessary.

Table 1-24 Reversed Property in a Chain-Pattern Table

START_NODE_ID	...	P<Id(:motherOf)>	...	R<Id(:fatherOf)>
Id(:alice)	...	Id(:john)	...	Id(:bob)
Id(:alice)	...	Id(:mary)	...	Id(:bob)
Id(:kathy)	...	Id(:bob)	...	Id(:henry)

Example 1-100 Creating a Chain-Pattern Table

The following example creates a chain-pattern table representing the grandfather chain using two occurrences of the :fatherOf property on an RDF graph named M1.

```
BEGIN
  SEM_APIS.BUILD_RESULT_TAB(
     result_tab_name    => ‘GRANDPA’
   , query_pattern_type => SEM_APIS.SPM_TYPE_PCN
   , rdf_graph_name     => 'M1'
   , key_string         => ' S :fatherOf :fatherOf '
```

```
    , prefixes            => ' PREFIX : <http://www.example.com#> '
    , network_owner       => 'RDFUSER'
    , network_name        => 'NET1'
  );
END;
/
```

The name, structure, and default indexes for a chain-pattern table may be described as follows:

- The name of a chain-pattern table is based on the following template: `<NETWORK_NAME>#RDF_XT$`**`PCN`**`_<MODEL_NAME>+__<SPM_NAME>`
- The `NUMBER` column, `START_NODE_ID`, stores the subject id or, if reversed, the object id, of the matching triple for the first property in the sequence of properties in the chain-pattern table.
- For each property's n-th occurrence in a chain-pattern table, the following columns are created for storing the numeric identifiers for the lexical values in a triple: (note that the `#n` suffix is used only if `n > 1`):
  - `NUMBER` column `G<Id(property)>` (or `G<Id(property)>#n`) for storing the named graph id
  - `NUMBER` column `P<Id(property)>` (or `P<Id(property)>#n`) or, if reversed, `R<Id(property)>` (or `R<Id(property)>#n`), for storing the object id or, if reversed, the subject id
  - (Optional) additional columns for internal use
- A nonunique index, named using the template `<NETWORK_NAME>#RDF_XX$`**`PCN`**`_<MODEL_NAME>__<SPM_NAME>`, is created on the `START_NODE_ID` column.
- Additionally, a nonunique index is created on each of the property columns.

1.8.2 Creating and Managing Result Tables

The following sections explain the steps for creating and managing result tables.

- Including Lexical Values in Result Tables
  You can also include lexical values for objects in result tables.
- Creating and Dropping Secondary Indexes on Result Tables
  You can create and drop secondary indexes on result tables.
- Dropping Result Tables
  You can drop a specific result table.
- In-Memory Result Tables
  Taking advantage of Oracle Database In-Memory, you can create in-memory result tables using the `INMEMORY=T` flag in the options parameter.
- Metadata for Result Tables
  You can use the RDF_SPM_INFO view to retrieve metadata information for the result tables defined on an RDF graph.

- Utility Subprogram for Computing Per-Subject Cardinality Aggregates for Individual Properties
  You can use the SEM_APIS.GATHER_SPM_INFO procedure to create and populate a table to store the per-subject cardinality information for each property in an RDF graph, based on its use as predicate of triples.
- Performing DML Operations on RDF Graphs with Result Tables
  All star-pattern, triple-pattern, and chain-pattern tables are automatically maintained for DML operations.
- Performing Bulk Load Operations on RDF Graphs with Result Tables
- Gathering Statistics on Result Tables
  Having up-to-date statistics on result tables is critical for good query performance.

1.8.2.1 Including Lexical Values in Result Tables

You can also include lexical values for objects in result tables.

Result tables include numeric identifiers for object values by default. Additionally, by storing the lexical values (RDF terms) in the SPM tables, retrieval of lexical values during SPARQL query processing can be made faster by avoiding the lookups involving joins with the `RDF_VALUE$` table.

If you choose to include lexical values for the subject or values of any of the properties stored in a result table, new columns for the lexical property values are added to the star-pattern and chain-pattern tables. Note that these columns correspond exactly to the columns with the same name in `RDF_VALUE$`. Specifically, when including lexical values for a non-reversed property into a result table, the following columns get added to the result table:

- `P<Id(property)>_VALUE_TYPE`
- `P<Id(property)>_VNAME_PREFIX`
- `P<Id(property)>_VNAME_SUFFIX`
- `P<Id(property)>_LITERAL_TYPE`
- `P<Id(property)>_LANGUAGE_TYPE`
- `P<Id(property)>_ORDER_NUM`
- `P<Id(property)>_ORDER_DATE`
- `P<Id(property)>_LONG_VALUE`

For reversed properties, the column names use `'R'` as the first character instead of the character `'P'`. Names for the additional columns added for including the lexical values for the subject (that is, corresponding to the numeric identifiers stored in the `START_NODE_ID` column), use the prefix `'S'`, instead of `P<Id(property)>` or `R<Id(property)>`.

The following example is a variation of Example 1-98, in that the lexical values for the subject and the reversed `:fatherOf` property are included. The `'+'` symbol is used to indicate that lexical values needed to be stored in the result table. Here, use of `'+S'` and `'+^:fatherOf'` in the `key_string` parameter causes the additional columns to get added for the subject and the (reversed) `:fatherOf` property, respectively.

Example 1-101 Including Lexical Values for the Subject and for the Reversed Property

```
BEGIN
  SEM_APIS.BUILD_RESULT_TAB(
     query_pattern_type => SEM_APIS.SPM_TYPE_SVP
```

```
    , result_tab_name    => 'FLHF'
    , rdf_graph_name     => 'M1'
    , key_string         => ' +S :fname :lname :height +^:fatherOf '
    , prefixes           => ' PREFIX : <http://www.example.com#> '
    , network_owner      => 'RDFUSER'
    , network_name       => 'NET1'
  );
END;
/
```

If a result table is already present, you can use the SEM_APIS.ALTER_RESULT_TAB subprogram to include lexical values for either the subject or any one of the properties by using the string `'ADD_S_VALUE'` or `'ADD_VALUE'`, respectively, as value for the command parameter. The following example results in inclusion of the lexical values for the `:lname property`. (The command `DROP_S_VALUE` or `DROP_VALUE`, not shown in this example, can be used to remove the lexical value columns for the subject or a property, respectively.)

Example 1-102 Altering a Star-Pattern Table to Add Lexical Values for a Property

```
BEGIN
  SEM_APIS.ALTER_RESULT_TAB(
      query_pattern_type => SEM_APIS.SPM_TYPE_SVP
    , result_tab_name    => 'FLHF'
    , rdf_graph_name     => 'M1'
    , command            => 'ADD_VALUE'
    , pred_name          => '<http://www.example.com#lname>'
    , network_owner      => 'RDFUSER'
    , network_name       => 'NET1'
  );
END;
/
```

1.8.2.2 Creating and Dropping Secondary Indexes on Result Tables

You can create and drop secondary indexes on result tables.

If for a given workload, accessing the content of a result table through access paths other than those already provided by the default indexes on the result table are needed, corresponding secondary (B+-tree) indexes may be created by using the SEM_APIS.CREATE_INDEX_ON_RESULT_TAB subprogram.

The following example shows creation of such an index, named `name_idx`, on the star-pattern table created in Example 1-101. The `key_string` parameter, `'2P 1P S'`, indicates that the key should be the (numeric id) value from the column corresponding to the second property in the table, namely, `:lname`, followed by that from the first property in the table, namely, `:fname`, followed by the subject (that is, the `START_NODE_ID` column). Note that the reference to the *n-th* property is always `<n>P` regardless whether the corresponding column name in the result table is of the form `P<Id(property)>` or `R<Id(property)>`.

If the lexical values for a property are included in the result table, then the index key may also include one or more of the columns that store the components of the lexical values. To refer to a component, use the form `<n><component-code>`, where `n` is 0 (for `START_NODE_ID`) or position of the target property, and the component code is determined based on the suffix of the included value component names as shown in Table 1-25

Table 1-25 Mapping from Suffix of Lexical Value Component Column Names to Component Code

Suffix of Lexical Value Component Column Name	Component Code
VALUE_TYPE	VT
VNAME_PREFIX	VP
VNAME_SUFFIX	VS
LITERAL_TYPE	LT
LANGUAGE_TYPE	LA
ORDER_NUM	VN
ORDER_DATE	VD

For example, the reference to `2VP` and `0VP` in the key `'2P 1P` *`2VP 0VP`* `S'` indicates the inclusion of the following two columns in the key at the respective positions:

1. The `<`*`column_name_for_the_2nd_property_of_the _SPM_table`*`>_VNAME_PREFIX` column
2. The `S_VNAME_PREFIX` column (where `S` corresponds to the zeroth column of the SPM table, that is , the `START_NODE_ID` column).

Example 1-103 Creating a Secondary (B+-tree) Index on a Result Table

```
SEM_APIS.CREATE_INDEX_ON_RESULT_TAB(
     index_name.         => 'name_idx'
   , query_pattern_type => SEM_APIS.SPM_TYPE_SVP
   , result_tab_name     => 'FLHF'
   , rdf_graph_name      => 'M1'
   , key_string          => ' 2P 1P S '
   , network_owner       => 'RDFUSER'
   , network_name        => 'NET1'
  );
END;
/
```

To drop any index created using this subprogram, use the SQL `DROP INDEX <index_name>` command. For example:

```
DROP INDEX name_idx;
```

1.8.2.3 Dropping Result Tables

You can drop a specific result table.

You can use the SEM_APIS.DROP_RESULT_TAB subprogram to drop a result table as shown in the following example.

Example 1-104 Dropping a Result Table

```
BEGIN
  SEM_APIS.DROP_RESULT_TAB(
     query_pattern_type => SEM_APIS.SPM_TYPE_SVP
   , result_tab_name     => 'FLHF'
   , rdf_graph_name      => 'M1'
```

```
    , network_owner      => 'RDFUSER'
    , network_name       => 'NET1'
  );
END;
/
```

Note that the use of the special string, '*', for the `result_tab_name` parameter, allows dropping all result tables of the type specified by the `query_pattern_type` parameter. To drop all the result tables, regardless of the type, use `SEM_APIS.SPM_TYPE_ALL` for the `query_pattern_type` parameter.

1.8.2.4 In-Memory Result Tables

Taking advantage of Oracle Database In-Memory, you can create in-memory result tables using the `INMEMORY=T` flag in the options parameter.

Generally, on-disk result tables are designed based on the commonly occurring patterns in the individual queries in a workload. If the result tables contain extra columns that are not needed for the query, it could incur disk scan overhead. If the query workload is not known or varying, building a result table with all properties could be a good choice. The in-memory columnar format ensures that only the necessary columns are accessed. Only one in-memory result table with all properties can be built and any other result tables are not allowed.

The in-memory result table with all properties can be built using `'INMEMORY=T'` as shown in the following example.

Example 1-105 Creating an In-memory Result Table

As a prerequisite, ensure that the table `M1_PRED_INFO` that is used in this example already exists. This table can be created using the SEM_APIS.GATHER_SPM_INFO subprogram.

```
BEGIN
SEM_APIS.BUILD_RESULT_TAB(
  rdf_graph_name    =>'M1',
  pred_info_tabname =>'M1_PRED_INFO',
  pred_name       =>NULL,
  options         =>' INMEMORY=T ',
  degree          =>2,
  network_owner     =>'RDFUSER',
  network_name      =>'NET1'
  );
END;
/
```

If a set of properties to access for all queries is known, an in-memory SVP table with a subset of all properties can be built by altering the SVP table built using the set as follows:

```
ALTER TABLE "MYNET#RDF_XT$SVP_M1+__SVP1" INMEMORY;
```

1.8.2.5 Metadata for Result Tables

You can use the RDF_SPM_INFO view to retrieve metadata information for the result tables defined on an RDF graph.

Table 1-26 Predicate Information Table Columns

Column Name	Type	Description
TABLE_NAME	VARCHAR2(128)	Name of the SPM table.
COLUMN_NAME	VARCHAR2(128)	Name of a column in the SPM table: either `START_NODE_ID`, or `P<id(property)> or R<id(property)>`.
COLUMN_ID	NUMBER	Position of the column in the SPM table's column list.
HASVALUES	NUMBER(1)	Indicates if in addition to the numeric identifiers for the values, their lexical values too are stored in the SPM table.
MODEL_ID	NUMBER	Numeric identifier of the RDF graph.
MODEL_NAME	VARCHAR2(128)	Name of the RDF graph.

1.8.2.6 Utility Subprogram for Computing Per-Subject Cardinality Aggregates for Individual Properties

You can use the SEM_APIS.GATHER_SPM_INFO procedure to create and populate a table to store the per-subject cardinality information for each property in an RDF graph, based on its use as predicate of triples.

The `P_VALUE_ID` column stores the numeric identifier corresponding to a property. For a reversed property, `P_VALUE_ID` stores the negative value of the id for the property.

This property cardinality table has the structure as shown in the following table. If `MAX_CNT > 1` for a given property, then that property is multi-valued, that is, for at least one of the subject resources, this property has been used as predicate for two or more distinct triples (that share the same subject and same predicate but has distinct objects).

Table 1-27 Predicate Information Table Columns

Column Name	Type	Description
P_VALUE_ID	NUMBER	The value id for this property. A negative value indicates *reversed* property.
PRED_NAME	VARCHAR2(4000)	The lexical value for this property.
MIN_CNT	NUMBER	The minimum of the per-subject cardinalities for this property.
MAX_CNT	NUMBER	The maximum of the per-subject cardinalities for this property.
MED_CNT	NUMBER	The median of the per-subject cardinalities for this property.
AVG_CNT	NUMBER	The average of the per-subject cardinalities for this property.
TOT_CNT	NUMBER	The total number of triples that have this property as predicate.

Table 1-27 (Cont.) Predicate Information Table Columns

Column Name	Type	Description
INCLUDE	VARCHAR2(30)	Not used.

For the sample RDF dataset (described in Types of Result Tables), the cardinality information is described in the following table.

Table 1-28 Sample Cardinality Information in the Predicate Table

P_VALUE_ID	PRED_NAME	MIN_CNT	MAX_CNT	MED_CNT	AVG_CNT	TOT_CNT	INCLUDE
Id(:fname)	:fname	1	1	…	…	4	...
Id(:lname)	:lname	1	1	…	…	4	...
Id(:height)	:height	1	1	…	…	4	...
Id(:email)	:email	1	2	…	…	4	...
Id(:fatherOf)	:fatherOf	1	2	…	…	3	...
Id(:motherOf)	:motherOf	1	2	…	…	3	...

A second procedure, SEM_APIS.BUILD_RESULT_TAB, creates and populates star-pattern, triple-pattern, and chain-pattern tables.

The following example illustrates creation of a set of result tables for an RDF graph with SEM_APIS.GATHER_SPM_INFO and SEM_APIS.BUILD_RESULT_TAB. These result tables are automatically used for SPARQL query execution. This example uses SEM_MATCH, but SPARQL queries executed through other APIs, such as those supported for Apache Jena or RDF server will also automatically use result tables.

Example 1-106 Creating Result Tables and Using the Tables in SPARQL Queries

```
SQL> set echo on pages 10000 numwidth 20 lines 200 long 10000
SQL> column s format a30
SQL> column fname format a5
SQL> column lname format a5
SQL> column height format a6
SQL> column email format a25
SQL> column nick format a10
SQL> column friend format a30
SQL> column state format a5

SQL> conn rdfuser/rdfuser
Connected.

SQL> -- create an RDF network
SQL> exec
sem_apis.create_rdf_network('tbs_rdf',network_owner=>'RDFUSER',network_name=>'
NET1');

PL/SQL procedure successfully completed.

SQL> --move the RDF_SPM$ table and indexes defined on it to the network's
tablespace
SQL> alter table NET1#RDF_SPM$ move tablespace tbs_rdf;
```

```
SQL> set serverout on;
SQL> begin
  2   for idx in (select index_name from sys.user_indexes where
table_name='NET1#RDF_SPM$') loop
  3     execute immediate 'alter index "' || idx.index_name || '" rebuild
tablespace TBS_RDF';
  4     sys.dbms_output.put_line('moved (rebuild) index: ' || idx.index_name);
  5   end loop;
  6 end;
  7 /
SQL> set serverout off;

SQL> -- create an RDF graph
SQL> exec
sem_apis.create_rdf_graph('M1',null,null,network_owner=>'RDFUSER',network_name
=>'NET1');

PL/SQL procedure successfully completed.

SQL> -- add some data: fname, lname, height, and nickName are single-valued;
email and friendOf are multi-valued
SQL> begin
  2    sem_apis.update_rdf_graph('M1',
  3       'PREFIX    : <http://www.example.com#>
  4        PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
  5        INSERT DATA {
  6          :john :fname "John" ; :lname "Brown" ; :height 72
  7          ; :email "john@email-example.com", "johnnyB@email-example.com"
  8          ; :nickName "Johnny B"
  9          ; :friendOf :ann
 10          ; :address [ :addrNum 20 ; :addrStreet "Elm
Street" ; :addrCityState [ :addrCity "Boston" ; :addrState "MA" ] ] .
 11          :ann  :fname "Ann" ; :lname "Green" ; :height 65
 12          ; :email "ann@email-example.com"
 13          ; :nickName "Annie"
 14          ; :friendOf :john, :bill
 15          ; :address [ :addrNum 10 ; :addrStreet "Main
Street" ; :addrCityState [ :addrCity "New York" ; :addrState "NY" ] ] .
 16          :bill :fname "Bill" ; :lname "Red" ; :height 70
 17          ; :email "bill@email-example.com"
 18          ; :nickName "Billy"
 19          ; :friendOf :ann, :jane
 20          ; :address [ :addrNum 5 ; :addrStreet "Peachtree
Street" ; :addrCityState [ :addrCity "Atlanta" ; :addrState "GA" ] ] .
 21          :jane :fname "Jane" ; :lname "Blue" ; :height 68
 22          ; :email "jane@email-example.com", "jane2@email-example.com"
 23          ; :friendOf :bill
 24          ; :address [ :addrNum 101 ; :addrStreet "Maple
Street" ; :addrCityState [ :addrCity "Chicago" ; :addrState "IL" ] ] .
 25        }'
 26       ,network_owner=>'RDFUSER'
 27       ,network_name=>'NET1');
 28  end;
 29  /

PL/SQL procedure successfully completed.
```

```
SQL> -- create a star-pattern table for single-valued
predicates :fname, :lname, :height
SQL> BEGIN
  2    SEM_APIS.BUILD_RESULT_TAB(
  3        query_pattern_type => SEM_APIS.SPM_TYPE_SVP
  4      , result_tab_name    => 'fnm_lnm_hght'
  5      , rdf_graph_name     => 'M1'
  6      , key_string         => ' :fname :lname :height '
  7      , prefixes           => ' PREFIX : <http://www.example.com#> '
  8      , degree             => 2
  9      , network_owner      => 'RDFUSER'
 10      , network_name       => 'NET1'
 11    );
 12  END;
 13  /

PL/SQL procedure successfully completed.

SQL> -- check the star-pattern table
SQL> select * from "NET1#RDF_XT$SVP_M1+__FNM_LNM_HGHT" order by start_node_id;

      START_NODE_ID G8337314745347241189 P8337314745347241189
G7644445801044650266 P7644445801044650266 G4791477124431525340
P4791477124431525340
-------------------- -------------------- --------------------
-------------------- -------------------- --------------------
--------------------
 1399946303865654932
2838435233532231409
5036507830384741776
7949294891880010615
 7024748068782994892
9071571320455459462
8802343394415720481
7603694794035016230
 8531245907959123227
5085904049929492 3
9011354822640550059
4318017261525689661
 8972322488425499169
3239737248730612593
6648986869806945928
2028730158517518732

4 rows selected.

SQL> -- create a chain-pattern table for :address/:addrCityState/:addrState
SQL> BEGIN
  2    SEM_APIS.BUILD_RESULT_TAB(
  3        query_pattern_type => SEM_APIS.SPM_TYPE_PCN
  4      , result_tab_name    => 'addr_state'
  5      , rdf_graph_name     => 'M1'
  6      , key_string         => ' S :address :addrCityState :addrState '
  7      , prefixes           => ' PREFIX : <http://www.example.com#> '
  8      , degree             => 2
```

```
  9       , network_owner      => 'RDFUSER'
 10       , network_name       => 'NET1'
 11     );
 12   END;
 13   /

PL/SQL procedure successfully completed.

SQL> -- check the chain-pattern table content
SQL> -- Note: Since generated blank node labels may differ from run to run,
the 3rd and 5th column values may vary as well
SQL> select * from "NET1#RDF_XT$PCN_M1+__ADDR_STATE" order by start_node_id,
3, 5, 7;

START_NODE_ID        G5055192271510902740 P5055192271510902740
G2282073771135796724 P2282073771135796724 G594560333771551504
P594560333771551504
-------------------- -------------------- --------------------
-------------------- -------------------- --------------------
--------------------
1399946303865654932                  6519232173603163724
2583525877732786353                   2028557412112123936
7024748068782994892                  5974521208853734660
3828178052943534859                   7995579594576433205
8531245907959123227                  7758805114187110754
6401534854183681859                   5359878998404290171
8972322488425499169                   875920943154203631
3729916732662692051                   4933462079191011078

4 rows selected.

SQL> -- create triple-pattern tables for :email and :friendOf
SQL> -- :email
SQL> BEGIN
  2     SEM_APIS.BUILD_RESULT_TAB(
  3         query_pattern_type => SEM_APIS.SPM_TYPE_MVP
  4       , result_tab_name    => null
  5       , rdf_graph_name     => 'M1'
  6       , key_string         => ' :email '
  7       , prefixes           => ' PREFIX : <http://www.example.com#> '
  8       , degree             => 2
  9       , network_owner      => 'RDFUSER'
 10       , network_name       => 'NET1'
 11     );
 12   END;
 13   /

PL/SQL procedure successfully completed.

SQL> -- check the triple-pattern table
SQL> select * from "NET1#RDF_XT$MVP_M1+_P2930492586059823454" order by
start_node_id;

       START_NODE_ID G2930492586059823454
P2930492586059823454
```

```
-------------------- --------------------
--------------------


 1399946303865654932
6100245385739701229


 7024748068782994892
2096397932624357828


 7024748068782994892
6480436012276020283


 8531245907959123227
1846003049324830366


 8531245907959123227
7834835188342349976


 8972322488425499169
7251371240613573863



6 rows selected.

SQL> -- :friendOf
SQL> BEGIN
  2    SEM_APIS.BUILD_RESULT_TAB(
  3        query_pattern_type => SEM_APIS.SPM_TYPE_MVP
  4      , result_tab_name    => null
  5      , rdf_graph_name     => 'M1'
  6      , key_string         => ' :friendOf '
  7      , prefixes           => ' PREFIX : <http://www.example.com#> '
  8      , degree             => 2
  9      , network_owner      => 'RDFUSER'
 10      , network_name       => 'NET1'
 11    );
 12  END;
 13  /

PL/SQL procedure successfully completed.

SQL> -- check the triple-pattern table
SQL> select * from "NET1#RDF_XT$MVP_M1+_P1285894645615718351" order by
start_node_id, 3;

       START_NODE_ID G1285894645615718351
P1285894645615718351
```

```
-------------------- --------------------
--------------------


 1399946303865654932
7024748068782994892


 1399946303865654932
8972322488425499169


 7024748068782994892
1399946303865654932


 8531245907959123227
8972322488425499169


 8972322488425499169
1399946303865654932


 8972322488425499169
8531245907959123227



6 rows selected.

SQL> -- gather optimizer statistics on result auxiliary tables
SQL> begin
  2    sem_perf.analyze_aux_tables(
  3       model_name=>'M1',
  4       network_owner=>'RDFUSER',
  5       network_name=>'NET1');
  6  end;
  7  /

PL/SQL procedure successfully completed.

SQL> -- Execute a SPARQL query that uses result tables
SQL> SELECT s, fname, lname, height, email, nick, friend, state
  2  FROM TABLE(SEM_MATCH(
  3  'PREFIX : <http://www.example.com#>
  4   SELECT *
  5   WHERE {
  6      ?s :fname ?fname
  7       ; :lname ?lname
  8       ; :height ?height
  9       ; :email ?email
 10       ; :nickName ?nick
 11       ; :friendOf ?friend
 12       ; :address/:addrCityState/:addrState ?state
```

```
 13    }'
 14   ,sem_models('M1')
 15   ,null,null,null,null
 16   ,' '
 17   ,null,null
 18   ,'RDFUSER','NET1'))
 19   ORDER BY 1,2,3,4,5,6,7,8;

S                               FNAME LNAME HEIGHT EMAIL
NICK       FRIEND
STATE

------------------------------ ----- ----- ------ -------------------------
---------- ------------------------------
-----

http://www.example.com#ann     Ann   Green 65     ann@email-example.com
Annie      http://www.example.com#bill
NY

http://www.example.com#ann     Ann   Green 65     ann@email-example.com
Annie      http://www.example.com#john
NY

http://www.example.com#bill    Bill  Red   70     bill@email-example.com
Billy      http://www.example.com#ann
GA

http://www.example.com#bill    Bill  Red   70     bill@email-example.com
Billy      http://www.example.com#jane
GA

http://www.example.com#john    John  Brown 72     john@email-example.com
Johnny B   http://www.example.com#ann
MA

http://www.example.com#john    John  Brown 72     johnnyB@email-example.com
Johnny B   http://www.example.com#ann
MA


6 rows selected.

SQL> -- See the relevant portion of the SQL translation showing the result
table usage.
SQL> --
SQL> -- This SQL evaluates 9 triple patterns with only 4 joins
SQL> -- instead of the 8 joins that would normally be required
SQL> -- without result tables.
SQL> --
SQL> -- The star-pattern table is used for :fname, :lname, :height.
SQL> -- triple-pattern tables are used for :email and :friendOf.
SQL> -- RDFM_M1 (view of RDF_LINK$ for RDF graph M1) is used for :nickName.
SQL> -- The chain-pattern table is used for the sequence
SQL> -- :address/:addrCityState/:addrStat
SQL> SELECT sys.dbms_lob.substr(
```

```
SEM_APIS.SPARQL_TO_SQL(
'PREFIX : <http://www.example.com#>
 SELECT *
 WHERE {
    ?s :fname ?fname
     ; :lname ?lname
     ; :height ?height
     ; :email ?email
     ; :nickName ?nick
     ; :friendOf ?friend
     ; :address/:addrCityState/:addrState ?state
 }'
,sem_models('M1')
,null,null,null
,' '
,null,null
,'RDFUSER','NET1'), 1004, 3377) AS SQL_TRANS_PORTION
FROM SYS.DUAL;

SQL_TRANS_PORTION


------------------------------------------------------------------------------
------------------------------------------------------------------------------
--------------------------------------------
SELECT SVP0.START_NODE_ID AS S$RDFVID,
SVP0.P7644445801044650266 AS LNAME$RDFVID,
MVP1.P1285894645615718351 AS FRIEND$RDFVID,
T4.CANON_END_NODE_ID AS NICK$RDFVID,
PCN0.P594560333771551504 AS STATE$RDFVID,
SVP0.P4791477124431525340 AS HEIGHT$RDFVID,
MVP0.P2930492586059823454 AS EMAIL$RDFVID,
SVP0.P8337314745347241189 AS FNAME$RDFVID,
SVP0.START_NODE_ID AS BGP$1
FROM (
SELECT * FROM "RDFUSER".NET1#RDFM_M1) T4,
"RDFUSER"."NET1#RDF_XT$SVP_M1+__FNM_LNM_HGHT" SVP0,
"RDFUSER"."NET1#RDF_XT$PCN_M1+__ADDR_STATE" PCN0,
"RDFUSER"."NET1#RDF_XT$MVP_M1+_P2930492586059823454" MVP0,
"RDFUSER"."NET1#RDF_XT$MVP_M1+_P1285894645615718351" MVP1
WHERE SVP0.P8337314745347241189 IS NOT NULL AND
SVP0.P7644445801044650266 IS NOT NULL AND
SVP0.P4791477124431525340 IS NOT NULL AND
T4.P_VALUE_ID = 2558054308995111125 AND
1=1 AND
1=1 AND
1=1 AND
SVP0.START_NODE_ID = MVP0.START_NODE_ID AND
SVP0.START_NODE_ID = T4.START_NODE_ID AND
SVP0.START_NODE_ID = MVP1.START_NODE_ID AND
SVP0.START_NODE_ID = PCN0.START_NODE_ID AND
1=1
```

```
1 row selected.
```

Example 1-107   Including Lexical Values in Result Tables

Example 1-106

```
SQL> conn rdfuser/rdfuser

SQL> -- Drop and recreate the FNM_LNM_HGHT SVP table, with in-line lexical
values for the :fname and :height properties.
SQL> -- Check metadata for the new result table to verify that HASVALUES=1
for the two properties whose lexical values are in-lined.

SQL> exec sem_apis.drop_result_tab(sem_apis.SPM_TYPE_SVP, ' fnm_lnm_hght ',
'm1', network_owner=>'rdfuser', network_name=>'net1');

PL/SQL procedure successfully completed.

SQL>

SQL> BEGIN
     SEM_APIS.BUILD_RESULT_TAB(
         query_pattern_type => SEM_APIS.SPM_TYPE_SVP
       , result_tab_name    => 'fnm_lnm_hght'
       , rdf_graph_name     => 'M1'
       , key_string         => ' S +:fname :lname +:height '
       , prefixes           => ' PREFIX : <http://www.example.com#> '
       , degree             => 2
       , network_owner      => 'RDFUSER'
       , network_name       => 'NET1'
     );
   END;
   /

PL/SQL procedure successfully completed.

SQL>
SQL> select * from net1#rdf_spm_info where table_name like
'%SVP%FNM_LNM_HGHT' order by table_name, column_id;

TABLE_NAME                              COLUMN_NAME
COLUMN_ID            HASVALUES MODEL_ID
MODEL_NAME

---------------------------------------- --------------------
-------------------- -------------------- --------
----------


NET1#RDF_XT$SVP_M1+__FNM_LNM_HGHT
START_NODE_ID                           1                    0        1
M1

NET1#RDF_XT$SVP_M1+__FNM_LNM_HGHT
P8337314745347241189                    3                    1        1
M1
```

```
NET1#RDF_XT$SVP_M1+__FNM_LNM_HGHT
P7644445801044650266                    5                    0        1
M1

NET1#RDF_XT$SVP_M1+__FNM_LNM_HGHT
P4791477124431525340                    7                    1        1
M1


4 rows selected.

SQL>
SQL> -- Drop and recreate the ADDR_STATE chain-pattern table, with in-line
lexical values for the :addrState property.
SQL> -- Check metadata for the new table to verify that HASVALUES=1 for
the :addrState property.

SQL> exec sem_apis.drop_result_tab(sem_apis.SPM_TYPE_PCN, ' addr_state ',
'm1', network_owner=>'rdfuser', network_name=>'net1');

PL/SQL procedure successfully completed.

SQL>
SQL> BEGIN
  2    SEM_APIS.BUILD_RESULT_TAB(
  3        query_pattern_type => SEM_APIS.SPM_TYPE_PCN
  4      , result_tab_name    => 'addr_state'
  5      , rdf_graph_name     => 'M1'
  6      , key_string         => ' S :address :addrCityState +:addrState '
  7      , prefixes           => ' PREFIX : <http://www.example.com#> '
  8      , degree             => 2
  9      , network_owner      => 'RDFUSER'
 10      , network_name       => 'NET1'
 11    );
 12  END;
 13  /

PL/SQL procedure successfully completed.

SQL>
SQL> select * from net1#rdf_spm_info where table_name like '%PCN%ADDR_STATE'
order by table_name, column_id;

TABLE_NAME                               COLUMN_NAME
COLUMN_ID            HASVALUES MODEL_ID
MODEL_NAME

---------------------------------------- --------------------
-------------------- -------------------- ---------
----------

NET1#RDF_XT$PCN_M1+__ADDR_STATE
START_NODE_ID                           1                    0        1
M1
```

```
NET1#RDF_XT$PCN_M1+__ADDR_STATE
P5055192271510902740                3                    0        1
M1

NET1#RDF_XT$PCN_M1+__ADDR_STATE
P2282073771135796724                5                    0        1
M1

NET1#RDF_XT$PCN_M1+__ADDR_STATE
P594560333771551504                 7                    1        1
M1


4 rows selected.

SQL>
SQL> -- Drop and recreate the triple-pattern table for the :email property
(id: 2930492586059823454), with in-line lexical values for the :email
property.
SQL> -- Check metadata for the new table to verify that HASVALUES=1 for
the :email property.

SQL> exec sem_apis.drop_result_tab(sem_apis.SPM_TYPE_MVP, '<http://
www.example.com#email>', 'm1', network_owner=>'rdfuser',
network_name=>'net1');

PL/SQL procedure successfully completed.

SQL> BEGIN
  2    SEM_APIS.BUILD_RESULT_TAB(
  3        query_pattern_type => SEM_APIS.SPM_TYPE_MVP
  4      , result_tab_name    => null
  5      , rdf_graph_name     => 'M1'
  6      , key_string         => ' +:email '
  7      , prefixes           => ' PREFIX : <http://www.example.com#> '
  8      , degree             => 2
  9      , network_owner      => 'RDFUSER'
 10      , network_name       => 'NET1'
 11    );
 12  END;
 13  /

PL/SQL procedure successfully completed.

SQL>
SQL> select * from net1#rdf_spm_info where table_name like
'%MVP%P2930492586059823454' order by table_name, column_id;

TABLE_NAME                               COLUMN_NAME
COLUMN_ID            HASVALUES MODEL_ID
MODEL_NAME

---------------------------------------- --------------------
-------------------- -------------------- --------
----------
```

```
NET1#RDF_XT$MVP_M1+_P2930492586059823454
START_NODE_ID                          1                    0        1
M1

NET1#RDF_XT$MVP_M1+_P2930492586059823454
P2930492586059823454                   3                    1        1
M1


2 rows selected.

SQL>
SQL> -- gather optimizer statistics on result auxiliary tables
SQL> begin
  2    sem_perf.analyze_aux_tables(
  3       model_name=>'M1',
  4       network_owner=>'RDFUSER',
  5       network_name=>'NET1');
  6  end;
  7  /

PL/SQL procedure successfully completed.

SQL>
SQL> -- Execute a SPARQL query that uses result tables
SQL> SELECT s, fname, lname, height, email, nick, friend, state
  2  FROM TABLE(SEM_MATCH(
  3  'PREFIX : <http://www.example.com#>
  4   SELECT *
  5   WHERE {
  6      ?s :fname ?fname
  7       ; :lname ?lname
  8       ; :height ?height
  9       ; :email ?email
 10       ; :nickName ?nick
 11       ; :friendOf ?friend
 12       ; :address/:addrCityState/:addrState ?state
 13   }'
 14  ,sem_models('M1')
 15  ,null,null,null,null
 16  ,' '
 17  ,null,null
 18  ,'RDFUSER','NET1'))
 19  ORDER BY 1,2,3,4,5,6,7,8;

S                              FNAME LNAME HEIGHT EMAIL
NICK       FRIEND
STATE

------------------------------ ----- ----- ------ -------------------------
---------- ------------------------------
-----

http://www.example.com#ann     Ann   Green 65     ann@email-example.com
Annie      http://www.example.com#bill
NY
```

```
http://www.example.com#ann     Ann   Green 65     ann@email-example.com
Annie      http://www.example.com#john
NY

http://www.example.com#bill    Bill  Red   70     bill@email-example.com
Billy      http://www.example.com#ann
GA

http://www.example.com#bill    Bill  Red   70     bill@email-example.com
Billy      http://www.example.com#jane
GA

http://www.example.com#john    John  Brown 72     john@email-example.com
Johnny B   http://www.example.com#ann
MA

http://www.example.com#john    John  Brown 72     johnnyB@email-example.com
Johnny B   http://www.example.com#ann
MA


6 rows selected.

SQL>
SQL> -- See the relevant portion of the SQL translation showing SPM table
usage including in-line lexical values.
SQL> --
SQL> -- The number of joins with the RDF_VALUE$ table (for looking up lexical
values) goes down from 8 to 4
SQL> -- because out of the 8 variables being projected, 4 -- fname, height,
email, state -- appear
SQL> -- with properties whose lexical values are present in-line in the
available result tables.
SQL> --
SQL> SELECT SEM_APIS.SPARQL_TO_SQL(
  2  'PREFIX : <http://www.example.com#>
  3   SELECT *
  4   WHERE {
  5      ?s :fname ?fname
  6       ; :lname ?lname
  7       ; :height ?height
  8       ; :email ?email
  9       ; :nickName ?nick
 10       ; :friendOf ?friend
 11       ; :address/:addrCityState/:addrState ?state
 12   }'
 13  ,sem_models('M1')
 14  ,null,null,null
 15  ,' '
 16  ,null,null
 17  ,'RDFUSER','NET1')
 18  FROM SYS.DUAL;

SEM_APIS.SPARQL_TO_SQL('PREFIX:<HTTP://WWW.EXAMPLE.COM#>SELECT*WHERE{?S:FNAME?
FN
```

```
-------------------------------------------------------------------------------
--

SELECT * FROM
(


SELECT … <omitted> …
FROM (SELECT … <omitted>
…


FROM
(


SELECT * FROM "RDFUSER".NET1#RDFM_M1)
T4,


"RDFUSER"."NET1#RDF_XT$SVP_M1+__FNM_LNM_HGHT"
SVP0,

"RDFUSER"."NET1#RDF_XT$PCN_M1+__ADDR_STATE"
PCN0,

"RDFUSER"."NET1#RDF_XT$MVP_M1+_P2930492586059823454"
MVP0,

"RDFUSER"."NET1#RDF_XT$MVP_M1+_P1285894645615718351"
MVP1

WHERE 1=1
AND


1=1
AND


1=1
AND


1=1
AND


SVP0.P8337314745347241189 IS NOT NULL
AND


SVP0.P7644445801044650266 IS NOT NULL
AND
```

```
SVP0.P4791477124431525340 IS NOT NULL
AND


T4.P_VALUE_ID = 2558054308995111125
AND


1=1
AND


1=1
AND


1=1
AND


SVP0.START_NODE_ID = MVP0.START_NODE_ID
AND


SVP0.START_NODE_ID = T4.START_NODE_ID
AND


SVP0.START_NODE_ID = MVP1.START_NODE_ID
AND


SVP0.START_NODE_ID = PCN0.START_NODE_ID
AND


1=1) R, "RDFUSER".NET1#RDF_VALUE$ V0, "RDFUSER".NET1#RDF_VALUE$ V1,
"RDFUSER".NET1#RDF_VALUE$ V2, "RDFUSER".NET1#RDF_VALUE$
V3

WHERE (1=1)  AND (R.S$RDFVID = V0.VALUE_ID) AND (R.LNAME$RDFVID =
V1.VALUE_ID) AND (R.FRIEND$RDFVID = V2.VALUE_ID) AND (R.NICK$RDFVID =
V3.VALUE_ID)

) WHERE
(1=1)





1 row selected.

SQL>
```

```
SQL> -- In addition to value projection. In-line lexical values
SQL> -- can be used to evaluate FILTER conditions.
SQL> -- The value for ?height can be taken directly from the
SQL> -- SVP table in this case.
SQL> SELECT s, height
  2  FROM TABLE(SEM_MATCH(
  3  'PREFIX : <http://www.example.com#>
  4   SELECT ?s ?height
  5   WHERE {
  6      ?s :fname ?fname
  7       ; :lname ?lname
  8       ; :height ?height
  9      FILTER (?height >= 72)
 10   }'
 11  ,sem_models('M1')
 12  ,null,null,null,null
 13  ,' '
 14  ,null,null
 15  ,'RDFUSER','NET1'))
 16  ORDER BY 1,2;

S
HEIGHT


------------------------------
------


http://www.example.com#john
72



1 row selected.

SQL>
SQL> -- The SQL translation shows in-line lexical value usage for ?height >=
72.
SQL> SELECT SEM_APIS.SPARQL_TO_SQL(
  2  'PREFIX : <http://www.example.com#>
  3   SELECT ?s ?height
  4   WHERE {
  5      ?s :fname ?fname
  6       ; :lname ?lname
  7       ; :height ?height
  8      FILTER (?height >= 72)
  9   }'
 10  ,sem_models('M1')
 11  ,null,null,null
 12  ,' '
 13  ,null,null
 14  ,'RDFUSER','NET1') AS SQL_TRANS
 15  FROM SYS.DUAL;

SQL_TRANS
```

```
------------------------------------------------------------------------------
--

SELECT * FROM
(

SELECT … <omitted>
…

FROM (SELECT …<omitted>
…

FROM "RDFUSER"."NET1#RDF_XT$SVP_M1+__FNM_LNM_HGHT"
SVP0

WHERE 1=1
AND

SVP0.P8337314745347241189 IS NOT NULL
AND

SVP0.P7644445801044650266 IS NOT NULL
AND

SVP0.P4791477124431525340 IS NOT NULL
AND

1=1
AND

1=1
AND

(SVP0.P4791477124431525340_ORDER_NUM >= to_number(72))) R,
"RDFUSER".NET1#RDF_VALUE$
V0

WHERE (1=1)  AND (R.S$RDFVID =
V0.VALUE_ID)

) WHERE
(1=1)
```

```
1 row selected.

SQL>
```

Example 1-108   Creating Secondary Indexes on Result Auxiliary Tables

The following example illustrates creation of secondary indexes on result auxiliary tables. Note that this example follows Example 1-106 and Example 1-107.

```
SQL>
SQL> conn rdfuser/rdfuser
Connected.
SQL>
SQL> -- create index on the ORDER_NUM (VN) component of the lexical value of
the :height property.
SQL> -- This component is stored as a column in the FNM_LNM_HGHT SVP table.
SQL> -- It holds the numeric value for RDF literals of numeric type.
SQL> -- Since the :height property is the 3rd property in the SVP table, it
is referred to using 3VN in the key_string argument below.

SQL> BEGIN
  2    SEM_APIS.CREATE_INDEX_ON_RESULT_TAB(
  3        index_name          => 'height_idx'
  4      , query_pattern_type => SEM_APIS.SPM_TYPE_SVP
  5      , result_tab_name    => 'fnm_lnm_hght'
  6      , rdf_graph_name     => 'M1'
  7      , key_string         => ' 3VN S '
  8      , degree             => 2
  9      , network_owner      => 'RDFUSER'
 10      , network_name       => 'NET1'
 11    );
 12  END;
 13  /

PL/SQL procedure successfully completed.

SQL>
SQL> -- EXPLAIN PLAN for the SPARQL query above involving "height >= 72"
shows use of this index for access.
SQL> EXPLAIN PLAN FOR
  2  SELECT s, height
  3  FROM TABLE(SEM_MATCH(
  4  'PREFIX : <http://www.example.com#>
  5   SELECT ?s ?height
  6   WHERE {
  7      ?s :fname ?fname
  8       ; :lname ?lname
  9       ; :height ?height
 10      FILTER (?height >= 72)
 11   }'
 12  ,sem_models('M1')
```

```
 13  ,null,null,null,null
 14  ,' '
 15  ,null,null
 16  ,'RDFUSER','NET1'))
 17  ORDER BY 1,2;

Explained.

SQL>
SQL> select plan_table_output from
table(dbms_xplan.display('plan_table',null,'basic +predicate'));

PLAN_TABLE_OUTPUT


-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
--------------------------------------------
Plan hash value:
3046664063




-------------------------------------------------------------------------------
-------

| Id  | Operation                       |
Name
|

-------------------------------------------------------------------------------
-------

|   0 | SELECT STATEMENT
|
|

|   1 |  SORT ORDER BY
|
|

|   2 |   NESTED LOOPS
|
|

|   3 |    NESTED LOOPS
|
|

|   4 |     VIEW
|
|

|*  5 |      TABLE ACCESS BY INDEX ROWID BATCHED|
```

```
NET1#RDF_XT$SVP_M1+__FNM_LNM_HGHT
|

|*  6 |        INDEX RANGE SCAN                  |
HEIGHT_IDX
|

|*  7 |      INDEX UNIQUE SCAN                   |
NET1#C_PK_VID
|

|   8 |     TABLE ACCESS BY INDEX ROWID          |
NET1#RDF_VALUE$
|

------------------------------------------------------------------------------
-------



Predicate Information (identified by operation
id):

---------------------------------------------------




   5 - filter("SVP0"."P8337314745347241189" IS NOT NULL
AND

              "SVP0"."P7644445801044650266" IS NOT NULL AND
"SVP0"."P4791477124431525340"

              IS NOT
NULL)


   6 - access("SVP0"."P4791477124431525340_ORDER_NUM">=72
AND

              "SVP0"."START_NODE_ID">0 AND
"SVP0"."P4791477124431525340_ORDER_NUM" IS
NOT


NULL)



filter("SVP0"."START_NODE_ID">0)


   7 -
```

```
access("R"."S$RDFVID"="V0"."VALUE_ID")



27 rows selected.

SQL>
SQL> select column_name, column_position from all_ind_columns where
index_name='HEIGHT_IDX' order by 2;

COLUMN_NAME                      COLUMN_POSITION
------------------------------    --------------------
P4791477124431525340_ORDER_NUM    1
START_NODE_ID                  2
```

1.8.2.7 Performing DML Operations on RDF Graphs with Result Tables

All star-pattern, triple-pattern, and chain-pattern tables are automatically maintained for DML operations.

- **Delete:** For delete operations, corresponding rows from the triple-pattern table are deleted. In star-pattern tables, the corresponding column value is set to null including value columns. In chain-pattern tables, rows that use the deleted triple are deleted to reflect the removal of a link in the chain.
- **Insert:** For insert operations, a new subject row or the corresponding column value is inserted into the triple-pattern table if it does not exist including value columns. For star-pattern and chain-pattern tables, a new subject row or the column value is inserted if the existing value is null. If a different value is inserted than the existing value, an error is raised for constraint violation for star-pattern table.

1.8.2.8 Performing Bulk Load Operations on RDF Graphs with Result Tables

When bulk-loading RDF data into an RDF graph, if any result tables are present for the graph, those will be truncated before loading the data and re-populated after the loading has been completed.

1.8.2.9 Gathering Statistics on Result Tables

Having up-to-date statistics on result tables is critical for good query performance.

You can call the SEM_PERF.ANALYZE_AUX_TABLES procedure to gather statistics for your result tables.

1.8.3 SPARQL Query Options for Result Tables

SPARQL queries will automatically use result tables if they are present.

An existing SPARQL workload does not need to change to take advantage of result tables. However, several new query options and optimizer hints can be used to fine-tune result table usage.

The following query options can be used in the options argument of `SEM_MATCH` or in the `SEM_FS_NS` prefix used by support for Apache Jena and RDF Server.

- `COST_BASED_SPM_OPT` – usage of result tables is determined by the query execution plan cost
- `DISABLE_SPM_OPT` – do not use result tables (star-pattern, triple-pattern, and chain-pattern)
- `DISABLE_SVP_OPT` – do not use star-pattern tables
- `DISABLE_PCN_OPT` – do not use chain-pattern tables
- `DISABLE_MVP_OPT` – do not use triple-pattern tables
- `DISABLE_SPM_VALUES_OPT` – do not use in-line lexical values in result tables for value projection or filter evaluation (star-pattern, triple-pattern, and chain-pattern)
- `DISABLE_SPM_VALUE_PROJ_OPT` – do not use in-line lexical values in result tables for value projection (star-pattern, triple-pattern, and chain-pattern)
- `MIN_SVP_CLUSTER_SIZE(n)` – only use the star-pattern table for star pattern clusters that reference at least n properties contained in the star-pattern table (n = 1 by default)
- `PREFER_PCN=T` – when a triple pattern can be evaluated using either a star-pattern or chain-pattern table, choose the chain-pattern table (the default behavior is to use the star-pattern table)

The following query optimizer hints can be used in `HINT0` hint strings, the options argument of `SEM_MATCH`, and the `SEM_FS_NS` prefix used by support for Apache Jena and RDF server.

- `ALL_SPM_HASH / ALL_SPM_NL` – use hash / nested-loop join for all joins with result tables (star-pattern, triple-pattern, and chain-pattern)
- `ALL_SVP_HASH / ALL_SVP_NL` – use hash / nested-loop join for all joins with star-pattern tables
- `ALL_MVP_HASH / ALL_MVP_NL` – use hash / nested-loop join for all joins with triple-pattern tables
- `ALL_PCN_HASH / ALL_PCN_NL` – use hash / nested-loop join for all joins with chain-pattern tables

1.8.4 Special Considerations when Using Result Tables

This section describes a few limitations to be considered when using result tables.

- Result tables are only supported for a single RDF graph. RDF graph collections and inferred graphs are not supported.
- Result tables are not supported on RDF networks that are using Oracle Label Security.
- Flashback queries are not supported with result tables.
- An RDF graph with result tables cannot be used as the destination RDF graph in a SEM_APIS.MERGE_RDF_GRAPHS operation.
- SPARQL queries that use GeoSPARQL functions or Oracle Text functions do not utilize result tables.
- Evaluation of + and * property path expressions does not utilize result tables.
- Result tables are not supported for `SEM_APIS.APPEND_SEM_NETWORK_DATA`, `SEM_APIS.MOVE_SEM_NETWORK_DATA` or `SEM_APIS.RESTORE_SEM_NETWORK_DATA` operations.

1.9 Using the SEM_APIS.SPARQL_TO_SQL Function to Query RDF Data

You can use the SEM_APIS.SPARQL_TO_SQL function as an alternative to the SEM_MATCH table function to query RDF data.

> **Note:**
>
> The SEM_APIS.SPARQL_TO_SQL function is supported only if Oracle JVM is enabled on your Oracle Autonomous Database Serverless deployments. To enable Oracle JVM, see Use Oracle Java in *Using Oracle Autonomous Database Serverless* for more information.

The SEM_APIS.SPARQL_TO_SQL function is provided as an alternative to the SEM_MATCH table function. It can be used by application developers to obtain the SQL translation for a SPARQL query. This is the same SQL translation that would be executed by SEM_MATCH. The resulting SQL translation can then be executed in the same way as any other SQL string (for example, with EXECUTE IMMEDIATE in PL/SQL applications or with JDBC in Java applications).

The first (`sparql_query`) parameter to SEM_APIS.SPARQL_TO_SQL specifies a SPARQL query string and corresponds to the query argument of SEM_MATCH. In this case, however, `sparql_query` is of type CLOB, which allows query strings longer than 4000 bytes (or 32K bytes with long VARCHAR enabled). All other parameters are exactly equivalent to the same arguments of SEM_MATCH (described in Using the SEM_MATCH Table Function to Query RDF Data). The SQL query string returned by SEM_APIS.SPARQL_TO_SQL will produce the same return columns as an execution of SEM_MATCH with the same arguments.

The following PL/SQL fragment is an example of using the SEM_APIS.SPARQL_TO_SQL function.

```
DECLARE
  c           sys_refcursor;
  sparql_stmt clob;
  sql_stmt    clob;
  x_value     varchar2(4000);
BEGIN
  sparql_stmt :=
    'PREFIX : <http://www.example.org/family/>
     SELECT ?x
     WHERE {
       ?x :grandParentOf ?y .
       ?x rdf:type :Male
     }';

  sql_stmt := sem_apis.sparql_to_sql(
                sparql_stmt,
                sem_models('family'),
                SEM_Rulebases('RDFS','family_rb'),
                null,
                null,
                ' PLUS_RDFT=VC ', null, null,
```

```
                'RDFUSER', 'NET1');

  open c for 'select x$rdfterm from(' || sql_stmt || ')';
  loop
    fetch c into x_value;
    exit when c%NOTFOUND;

    dbms_output.put_line('x_value: ' || x_value);
  end loop;
  close c;

END;
/
```


1.9.1 Using Bind Variables with SEM_APIS.SPARQL_TO_SQL

The SEM_APIS.SPARQL_TO_SQL function allows the use of PL/SQL and JDBC bind variables. This is possible because the SQL translation returned from SEM_APIS.SPARQL_TO_SQL does not involve an ANYTYPE table function invocation. The basic strategy is to transform simple SPARQL BIND clauses into either JDBC or PL/SQL bind variables when the `USE_BIND_VAR=PLSQL` or `USE_BIND_VAR=JDBC` query option is specified. A simple SPARQL BIND clause is one with the form `BIND (<constant> AS ?var)`.

With the bind variable option, the SQL translation will contain two bind variables for each transformed SPARQL query variable: one for the value ID, and one for the RDF term string. An RDF term value can be substituted for a SPARQL query variable by specifying the value ID (from RDF_VALUE$ table) as the first bind value and the RDF term string as the second bind value. The value ID for a bound-in RDF term is required for performance reasons. The typical workflow would be to look up the value ID for an RDF term from the RDF_VALUE$ table (or with SEM_APIS.RES2VID) and then bind the ID and RDF term into the translated SQL.

Multiple query variables can be transformed into bind variables in a single query. In such cases, bind variables in the SQL translation will appear in the same order as the SPARQL BIND clauses appear in the SPARQL query string. That is, the (id, term) pair for the first BIND clause should be bound first, and the (id, term) pair for the second BIND clause should be bound second.

The following example shows the use of bind variables for SEM_APIS.SPARQL_TO_SQL from a PL/SQL block. A dummy bind variable `?n` is declared..

```
DECLARE
  sparql_stmt clob;
  sql_stmt    clob;
  cur         sys_refcursor;
  vid         number;
  term        varchar2(4000);
  c_val       varchar2(4000);
BEGIN
  -- Add a dummy bind clause in the SPARQL statement
  sparql_stmt := 'PREFIX : <http://www.example.org/family/>
                  SELECT ?c WHERE {
                  BIND("" as ?s)
```

```
                  ?s :parentOf ?c }';
  -- Get the SQL translation for SPARQL statement
  sql_stmt := sem_apis.sparql_to_sql(
                sparql_stmt,
                sem_models('family'),
                SEM_Rulebases('RDFS','family_rb'),
                null,
                null,' USE_BIND_VAR=PLSQL PLUS_RDFT=VC ', null, null,
                'RDFUSER', 'NET1');

  -- Execute with <http://www.example.org/family/Martha>
  term := '<http://www.example.org/family/Martha>';
  vid := sem_apis.res2vid('RDFUSER.NET1#RDF_VALUE$',term);

  dbms_output.put_line(chr(10)||'?s='||term);
  open cur for 'select c$rdfterm from('|| sql_stmt || ')' using vid,term;
  loop
    fetch cur into c_val;
    exit when cur%NOTFOUND;
    dbms_output.put_line('|-->?c='||c_val);
  end loop;
  close cur;

  -- Execute with <http://www.example.org/family/Sammy>
  term := '<http://www.example.org/family/Sammy>';
  vid := sem_apis.res2vid('RDFUSER.NET1#RDF_VALUE$',term);

  dbms_output.put_line(chr(10)||'?s='||term);
  open cur for 'select c$rdfterm from('|| sql_stmt || ')' using vid,term;
  loop
    fetch cur into c_val;
    exit when cur%NOTFOUND;
    dbms_output.put_line('|-->?c='||c_val);
  end loop;
  close cur;

END;
/
```

The following example shows the use of bind variables from Java for SEM_APIS.SPARQL_TO_SQL. In this case, the hint `USE_BIND_VAR=JDBC` is used.

```
public static void sparqlToSqlTest() {

    try {
        // Get connection
        Connection conn=DriverManager.getConnection(

"jdbc:oracle:thin:@localhost:1521:orcl","testuser","testuser");

        String sparqlStmt =
            "PREFIX : http://www.example.org/family/ \n" +
            "SELECT ?c WHERE {  \n" +
            "  BIND(\"\" as ?s) \n" +
            "  ?s :parentOf ?c      \n" +
```

```
            "}";

        // Get SQL translation of SPARQL statement
        // through sem_apis.sparql_to_sql
        OracleCallableStatement ocs =
(OracleCallableStatement)conn.prepareCall(
            "begin" +
            "  ? := " +
            "    sem_apis.sparql_to_sql('" +
            "      "+sparqlStmt+"'," +
            "      sem_models('family')," +
            "      SEM_Rulebases('RDFS','family_rb')," +
            "      null,null," +
            "   ' USE_BIND_VAR=JDBC PLUS_RDFT=VC " +
            " ',null,null,'RDFUSER','NET1');" +
            "end;");
        ocs.registerOutParameter(1,Types.VARCHAR);
        ocs.execute();
        String sqlStmt = ocs.getString(1);
        ocs.close();

        // Set up statement to look up value ids
        OracleCallableStatement ocsVid =
(OracleCallableStatement)conn.prepareCall(
            "begin" +
            "  ? := sem_apis.res2vid(?,?);" +
            "end;");

        // Execute SQL setting values for a bind variable
        PreparedStatement stmt=conn.prepareStatement(sqlStmt);

        // Look up value id for first value
        long valueId = 0;
        String term = "<http://www.example.org/family/Martha>";
        ocsVid.registerOutParameter(1,Types.NUMERIC);
        ocsVid.setString(2,"RDFUSER.NET1#RDF_VALUE$");
        ocsVid.setString(3,term);
        ocsVid.execute();
        valueId = ocsVid.getLong(1);

        stmt.setLong(1, valueId);
        stmt.setString(2, term);
        ResultSet rs=stmt.executeQuery();

        // Print results
        System.out.println("\n?s="+term);
        while(rs.next()) {
            System.out.println("|-->?c=" + rs.getString("c$rdfterm"));
        }
        rs.close();

        // Execute the same query for a different URI
        // Look up value id for next value
        valueId = 0;
        term = "<http://www.example.org/family/Sammy>";
        ocsVid.registerOutParameter(1,Types.NUMERIC);
```

```
                ocsVid.setString(2,"RDFUSER.NET1#RDF_VALUE$");
                ocsVid.setString(3,term);
                ocsVid.execute();
                valueId = ocsVid.getLong(1);

                stmt.setLong(1, valueId);
                stmt.setString(2, term);
                rs=stmt.executeQuery();

                // Print results
                System.out.println("\n?s="+term);
                while(rs.next()) {
                    System.out.println("|-->?c=" + rs.getString("c$rdfterm"));
                }
                rs.close();

                stmt.close();
                ocsVid.close();
                conn.close();

            } catch (SQLException e) {
                e.printStackTrace();
            }
        }
```

1.9.2 SEM_MATCH and SEM_APIS.SPARQL_TO_SQL Compared

The SEM_APIS.SPARQL_TO_SQL function avoids some limitations that are inherent in the SEM_MATCH table function due to its use of the rewritable table function interface. Specifically, SEM_APIS.SPARQL_TO_SQL adds the following capabilities.

- SPARQL query string arguments larger than 4000 bytes (32K bytes with long varchar support) can be used.
- The plain SQL returned from SEM_APIS.SPARQL_TO_SQL can be executed against read-only databases.
- The plain SQL returned from SEM_APIS.SPARQL_TO_SQL can support PL/SQL and JDBC bind variables.

SEM_MATCH, however, provides some unique capabilities that are not possible with SEM_APIS.SPARQL_TO_SQL..

- Support for projection optimization: If only the VAR$RDFVID column of a projected variable is selected from the SEM_MATCH invocation, the RDF_VALUE$ join for this variable will be avoided.
- Support for advanced features that require the procedural start-fetch-close table function execution: `SERVICE_JPDWN=T` and `OVERLOADED_NL=T` options with SPARQL SERVICE.
- The ability to execute queries interactively with tools like SQL*Plus.

1.10 Using the SEM_APIS.GET_SQL Function and SEM_SQL SQL Macro to Query RDF Data

You can use the SEM_APIS.GET_SQL function as an alternative to the SEM_MATCH table function to query RDF data.

It can be used by application developers to obtain the SQL translation for a SPARQL query. The resulting SQL translation can then be executed using SEM_SQL SQL Macro. The SEM_APIS.GET_SQL has exactly the same signature as SEM_APIS.SPARQL_TO_SQL function.

The following PL/SQL fragment is an example of using the SEM_APIS.GET_SQL function and SEM_SQL SQL Macro:

```
SQL> EXECUTE SEM_APIS.GET_SQL('SELECT ?s  ?o { ?s <http://www.w3.org/
1999/02/22-rdf-syntax-ns#type> ?o }',
sem_models('m1'),null,null,null,'
',null,null,network_owner=>'RDFUSER',network_name=>'MYNET');

PL/SQL procedure successfully completed.

SQL> SELECT count(s), count(o) FROM SEM_SQL();
            COUNT(S)             COUNT(O)
-------------------- --------------------
                   3                    3
1 row selected.

SQL> SELECT * FROM SEM_SQL() ORDER BY s,o;
S                                           S$RDFVID
------------------------------ --------------------
S$_PREFIX                      S$_SUFFIX                      S$RDFVTYP
------------------------------ ------------------------------ ----------
S$RDFCLOB            S$RDFLTYP                                S$RDFLANG
-------------------- ---------------------------------------- ----------
O                                           O$RDFVID
------------------------------ --------------------
O$_PREFIX                      O$_SUFFIX                      O$RDFVTYP
------------------------------ ------------------------------ ----------
O$RDFCLOB            O$RDFLTYP                                O$RDFLANG
-------------------- ---------------------------------------- ----------
          SEM$ROWNUM
--------------------
John                            4802682235912431956
John                                                          URI

OracleHQEmployee                9022701012979055032
OracleHQEmployee                                              URI
                                                                       1

Matt                            5972784495178428863
Matt                                                          URI

OracleHQEmployee                9022701012979055032
OracleHQEmployee                                              URI
                                                                       1
```

```
Sue                             8947116472173989398
Sue                                                           URI

OracleHQEmployee                9022701012979055032
OracleHQEmployee                                              URI
                                                                       1

3 rows selected.
```

Application developers can utilize SEM_SQL SQL Macro to run any translated query stored in some other tables using `RDF$S2S_SQL$` table and SEM_APIS.SEM_SQL_COMPILE to compile the SQL in the table as shown in the following example. This will save query translation time from SPARQL to SQL. Note that before using SEM_SQL for the first time, you must execute SEM_APIS.CREATE_SEM_SQL.

```
SQL> CREATE TABLE sql_tab(id int, s2s_sql clob);

Table created.

SQL> DECLARE
  2    sql_stmt    CLOB;
  3  BEGIN
  4    sql_stmt := sem_apis.SPARQL_TO_SQL('SELECT ?s  ?o { ?s <http://
www.w3.org/1999/02/22-rdf-syntax-ns#type> ?o }',
                       sem_models('m1'),null,null,null,'
',null,null,network_owner=>'RDFUSER2',network_name=>'MYNET');
  5    EXECUTE IMMEDIATE 'INSERT INTO sql_tab VALUES (1, :1)' USING sql_stmt;
  6
  7    sql_stmt := SEM_APIS.SPARQL_TO_SQL('SELECT ?s ?p ?o { ?s ?p ?o }',

sem_models('m1'),null,null,null,null,null,null,network_owner=>'RDFUSER2',netwo
rk_name=>'MYNET');
  8    EXECUTE IMMEDIATE 'INSERT INTO sql_tab VALUES (2, :1)' USING sql_stmt;
  9    EXECUTE IMMEDIATE 'commit';
 10  END;
 11  /

PL/SQL procedure successfully completed.

SQL> truncate table RDF$S2S_SQL$;

Table truncated.

SQL> INSERT INTO RDF$S2S_SQL$ SELECT s2s_sql FROM sql_tab WHERE id=1;

1 row created.

SQL> EXEC SEM_APIS.SEM_SQL_COMPILE;

PL/SQL procedure successfully completed.

SQL> SELECT  count(s), count(o) FROM sem_sql();
            COUNT(S)             COUNT(O)
-------------------- --------------------
                   3                    3
```

```
1 row selected.

SQL> SELECT  * FROM sem_sql() ORDER BY s,o;
S                                           S$RDFVID
------------------------------ --------------------
S$_PREFIX                      S$_SUFFIX                      S$RDFVTYP
------------------------------ ------------------------------ ----------
S$RDFCLOB            S$RDFLTYP                                S$RDFLANG
-------------------- ---------------------------------------- ----------
O                                           O$RDFVID
------------------------------ --------------------
O$_PREFIX                      O$_SUFFIX                      O$RDFVTYP
------------------------------ ------------------------------ ----------
O$RDFCLOB            O$RDFLTYP                                O$RDFLANG
-------------------- ---------------------------------------- ----------
          SEM$ROWNUM
--------------------
John                            4802682235912431956
John                                                          URI
OracleHQEmployee                9022701012979055032
OracleHQEmployee                                              URI
                   1
Matt                            5972784495178428863
Matt                                                          URI
OracleHQEmployee                9022701012979055032
OracleHQEmployee                                              URI
                   1
Sue                             8947116472173989398
Sue                                                           URI
OracleHQEmployee                9022701012979055032
OracleHQEmployee                                              URI
                   1

3 rows selected.

SQL> TRUNCATE TABLE RDF$S2S_SQL$;

Table truncated.

SQL> INSERT INTO RDF$S2S_SQL$ SELECT s2s_sql FROM sql_tab where id=2;

1 row created.

SQL> EXEC sem_apis.sem_sql_compile;
PL/SQL procedure successfully completed.

SQL> SELECT  count(*) from sem_sql();

            COUNT(*)
--------------------
                  26
1 row selected.

SQL> SELECT  * FROM sem_sql() ORDER BY s,p,o;
S                                           S$RDFVID
------------------------------ --------------------
```

```
S$_PREFIX                      S$_SUFFIX                      S$RDFVTYP
------------------------------ ------------------------------ ----------
S$RDFCLOB            S$RDFLTYP                                S$RDFLANG
-------------------- ---------------------------------------- ----------
P                                           P$RDFVID
------------------------------ --------------------
P$_PREFIX                      P$_SUFFIX                      P$RDFVTYP
------------------------------ ------------------------------ ----------
P$RDFCLOB            P$RDFLTYP                                P$RDFLANG
-------------------- ---------------------------------------- ----------
O                                           O$RDFVID
------------------------------ --------------------
O$_PREFIX                      O$_SUFFIX                      O$RDFVTYP
------------------------------ ------------------------------ ----------
O$RDFCLOB            O$RDFLTYP                                O$RDFLANG
-------------------- ---------------------------------------- ----------
          SEM$ROWNUM
--------------------
John                            4802682235912431956
John                                                          URI
age                             7369467453923552448
age                                                           URI
35                              9085530268529116130
35                                                            LIT
                     http://www.w3.org/2001/XMLSchema#decimal
                   1
John                            4802682235912431956
John                                                          URI
email                           6480734238761529200
email                                                         URI
john2@oracle.com                5315621098565335765
john2@oracle.com                                              LIT
                   1
John                            4802682235912431956
John                                                          URI
foaf                            2289371774016051690
foaf                                                          URI
Matt                            5972784495178428863
Matt                                                          URI
                   1
John                            4802682235912431956
John                                                          URI
http://www.w3.org/1999/02/22-r   834132227519661324
df-syntax-ns#type
http://www.w3.org/1999/02/22-r type                           URI
df-syntax-ns#
OracleHQEmployee                9022701012979055032
OracleHQEmployee                                              URI
                   1
John                            4802682235912431956
John                                                          URI
mbox                            5760688889368728142
mbox                                                          URI
john@oracle.com                 1322012223731379319
john@oracle.com                                               LIT
                   1
```

```
John                             4802682235912431956
John                                                           URI
name                             6027014909707307188
name                                                           URI
John Doe                         3287391926372438447
John Doe                                                       LIT
                   1
John                             4802682235912431956
John                                                           URI
nick                             4608123542649301902
nick                                                           URI
JD                               8942401707893765892
JD                                                             LIT
                   1
Matt                             5972784495178428863
Matt                                                           URI

age                              7369467453923552448
age                                                            URI

40                               1809238195348668799
40                                                             LIT
                     http://www.w3.org/2001/XMLSchema#decimal
                   1
Matt                             5972784495178428863
Matt                                                           URI
email                            6480734238761529200
email                                                          URI
matt2@oracle.com                 5816699135852471804
matt2@oracle.com                                               LIT
                   1
Matt                             5972784495178428863
Matt                                                           URI
foaf                             2289371774016051690
foaf                                                           URI
Su                               7425194847458329079
Su                                                             URI
                   1
Matt                             5972784495178428863
Matt                                                           URI
http://www.w3.org/1999/02/22-r    834132227519661324
df-syntax-ns#type
http://www.w3.org/1999/02/22-r type                            URI
df-syntax-ns#
OracleHQEmployee                 9022701012979055032
OracleHQEmployee                                               URI
                   1
Matt                             5972784495178428863
Matt                                                           URI
mbox                             5760688889368728142
mbox                                                           URI
matt@oracle.com                  1674614553190527316
matt@oracle.com                                                LIT
                   1
Matt                             5972784495178428863
Matt                                                           URI
```

```
name                            6027014909707307188
name                                                          URI
Matt Adams                      1025319037763704306
Matt Adams                                                    LIT
                   1
Matt                            5972784495178428863
Matt                                                          URI
teleCommFrom                     493206824495339087
teleCommFrom                                                  URI
teleCommLoc1                    4570292005318753230
teleCommLoc1                                                  URI
                   1
Sue                             8947116472173989398
Sue                                                           URI

age                             7369467453923552448
age                                                           URI

26                              4033985797457567386
26                                                            LIT
                     http://www.w3.org/2001/XMLSchema#decimal
                   1
Sue                             8947116472173989398
Sue                                                           URI
email                           6480734238761529200
email                                                         URI
sue2@oracle.com                 5229415107273694944
sue2@oracle.com                                               LIT
                   1
Sue                             8947116472173989398
Sue                                                           URI
http://www.w3.org/1999/02/22-r   834132227519661324
df-syntax-ns#type
http://www.w3.org/1999/02/22-r type                           URI
df-syntax-ns#
OracleHQEmployee                9022701012979055032
OracleHQEmployee                                              URI
                   1
Sue                             8947116472173989398
Sue                                                           URI
mbox                            5760688889368728142
mbox                                                          URI
sue@oracle.com                    31820890332196705
sue@oracle.com                                                LIT
                   1
Sue                             8947116472173989398
Sue                                                           URI
nick                            4608123542649301902
nick                                                          URI
Su                              4914588660956121377
Su                                                            LIT
                   1
Sue                             8947116472173989398
Sue                                                           URI
teleCommFrom                     493206824495339087
teleCommFrom                                                  URI
```

```
teleCommLoc2                   1084777556269608129
teleCommLoc2                                                  URI
                   1
email                          6480734238761529200
email                                                         URI
http://www.w3.org/2002/07/owl#  1982040897380465245
equivalentProperty
http://www.w3.org/2002/07/owl# equivalentProperty             URI
mbox                           5760688889368728142
mbox                                                          URI
                   1
teleCommLoc1                   4570292005318753230
teleCommLoc1                                                  URI
city                           3506365445274213635
city                                                          URI
NYC                            6275600577248419523
NYC                                                           URI
                   1
teleCommLoc1                   4570292005318753230
teleCommLoc1                                                  URI
state                          4843125665925023053
state                                                         URI
NY                              917887745150543696
NY                                                            URI
                   1
teleCommLoc1                   4570292005318753230
teleCommLoc1                                                  URI
zip                             627678011281517369
zip                                                           URI
10101                          9180109679895673868
10101                                                         LIT
                   1
teleCommLoc2                   1084777556269608129
teleCommLoc2                                                  URI
state                          4843125665925023053
state                                                         URI
NH                             3642103339971966862
NH                                                            URI
                   1
teleCommLoc2                   1084777556269608129
teleCommLoc2                                                  URI
zip                             627678011281517369
zip                                                           URI
03060                          2914451030353375942
03060                                                         LIT
                   1
26 rows selected.
```

1.11 Loading and Exporting RDF Data

You can load RDF data into an RDF graph in the database and export that data from the database into a staging table.

To load RDF data into an RDF graph, use one or more of the following options:

- Bulk load or append data into the RDF graph from a staging table, with each row containing the three components -- subject, predicate, and object -- of an RDF triple and optionally a named graph. This is explained in Bulk Loading RDF Data Using a Staging Table.

  This is the fastest option for loading large amounts of data.
- Load data into the application table using SQL INSERT statements that call the SDO_RDF_TRIPLE_S constructor, which results in the corresponding RDF triple, possibly including a graph name, to be inserted into the RDF data store, as explained in Loading RDF Data Using INSERT Statements.

  This option is convenient for loading small amounts of data
- Load data into the RDF graph with SPARQL Update statements executed through SEM_APIS.UPDATE_RDF_GRAPH, as explained in Support for SPARQL Update Operations on an RDF Graph.

  This option is convenient for loading small amounts of data, and can also be used to load larger amounts of data through LOAD statements.
- Load data into the RDF graph using the Apache Jena-based Java API, which is explained in RDF Graph Support for Apache Jena.

  This option provides several ways to load both small and large amounts of data, and it supports many different RDF serialization formats.

> **Note:**
>
> Unicode data in the staging table should be escaped as specified in WC3 N-Triples format (http://www.w3.org/TR/rdf-testcases/#ntriples). You can use the SEM_APIS.ESCAPE_RDF_TERM function to escape Unicode values in the staging table. For example:
>
> ```
> create table esc_stage_tab(rdf$stc_sub, rdf$stc_pred, rdf$stc_obj);
>
> insert /*+ append nologging parallel */ into esc_stage_tab
> (rdf$stc_sub, rdf$stc_pred, rdf$stc_obj)
> select sem_apis.escape_rdf_term(rdf$stc_sub, options=>’ UNI_ONLY=T
> '), sem_apis.escape_rdf_term(rdf$stc_pred, options=>’ UNI_ONLY=T '),
> sem_apis.escape_rdf_term(rdf$stc_obj, options=>’ UNI_ONLY=T ')
> from stage_tab;
> ```

To export RDF data, that is, to retrieve RDF data from Oracle Database where the results are in N-Triple or N-Quad format that can be stored in a staging table, use the SQL queries described in Exporting RDF Data.

> **Note:**
>
> Effective with Oracle Database Release 12.1, you can export and import a RDF network using the full database export and import features of the Oracle Data Pump utility, as explained in Exporting or Importing an RDF Network Using Oracle Data Pump.

1.11.1 Bulk Loading RDF Data Using a Staging Table

You can load RDF data (and optionally associated non-RDF data) in bulk using a staging table. Call the SEM_APIS.LOAD_INTO_STAGING_TABLE procedure (described in SEM_APIS Package Subprograms) to load the data, and you can have during the load operation to check for syntax correctness. Then, you can call the SEM_APIS.BULK_LOAD_RDF_GRAPH procedure to load the data into the RDF store from the staging table. (If the data was not parsed during the load operation into the staging table, you must specify the `PARSE` keyword in the `flags` parameter when you call the SEM_APIS.BULK_LOAD_RDF_GRAPH procedure.)

The following example shows the format for the staging table, including all required columns and the required names for these columns, plus the optional RDF$STC_graph column which must be included if one or more of the RDF triples to be loaded include a graph name:

```
CREATE TABLE stage_table (
                     RDF$STC_sub varchar2(4000) not null,
                     RDF$STC_pred varchar2(4000) not null,
                     RDF$STC_obj varchar2(4000) not null,
                     RDF$STC_graph varchar2(4000)
);
```

If you also want to load non-RDF data, specify additional columns for the non-RDF data in the CREATE TABLE statement. The non-RDF column names must be different from the names of the required columns. The following example creates the staging table with two additional columns (SOURCE and ID) for non-RDF attributes.

```
CREATE TABLE stage_table_with_extra_cols (
                     source VARCHAR2(4000),
                     id NUMBER,
                     RDF$STC_sub varchar2(4000) not null,
                     RDF$STC_pred varchar2(4000) not null,
                     RDF$STC_obj varchar2(4000) not null,
                     RDF$STC_graph varchar2(4000)
);
```

> **Note:**
>
> For either form of the CREATE TABLE statement, you may want to add the COMPRESS clause to use table compression, which will reduce the disk space requirements and may improve bulk-load performance.

Both the invoker and the network owner user must have the following privileges: SELECT privilege on the staging table, and INSERT privilege on the application table.

See also the following:

- Loading the Staging Table

- Recording Event Traces During Bulk Loading

1.11.1.1 Loading the Staging Table

You can load RDF data into the staging table, as a preparation for loading it into the RDF store, in several ways. Some of the common ways are the following:

- Loading N-Triple Format Data into a Staging Table Using SQL*Loader
- Loading N-Quad Format Data into a Staging Table Using an External Table

1.11.1.1.1 Loading N-Triple Format Data into a Staging Table Using SQL*Loader

You can use the SQL*Loader utility to parse and load RDF data into a staging table. If you installed the demo files from the Oracle Database Examples media (see *Oracle Database Examples Installation Guide*), a sample control file is available at `$ORACLE_HOME/md/demo/network/rdf_demos/bulkload.ctl`. You can modify and use this file if the input data is in N-Triple format.

Objects longer than `NETWORK_MAX_STRING_SIZE` bytes cannot be loaded. If you use the sample SQL*Loader control file, triples (rows) containing such long values will be automatically rejected and stored in a SQL*Loader "bad" file. However, you can load these rejected rows by inserting them into the application table using SQL INSERT statements (see Loading RDF Data Using INSERT Statements).

1.11.1.1.2 Loading N-Quad Format Data into a Staging Table Using an External Table

You can use an Oracle external table to load N-Quad format data (extended triple having four components) into a staging table, as follows:

1. Call the SEM_APIS.CREATE_SOURCE_EXTERNAL_TABLE procedure to create an external table, and then use the SQL STATEMENT ALTER TABLE to alter the external table to include the relevant input file name or names. You must have READ and WRITE privileges for the directory object associated with folder containing the input file or files.
2. Ensure the network owner has SELECT and INSERT privileges on the external table.

   If the network owner is invoking the routine to populate the staging table and then loading from the staging table, then ensure that the owner has SELECT privilege on the external table and both INSERT and SELECT privileges on the staging table.
3. Call the SEM_APIS.LOAD_INTO_STAGING_TABLE procedure to populate the staging table.
4. After the loading is finished, issue a COMMIT statement to complete the transaction.

Example 1-109   Using an External Table to Load a Staging Table

```
-- Create a source external table (note: table names are case sensitive)
BEGIN
  sem_apis.create_source_external_table(
    source_table    => 'stage_table_source'
   ,def_directory   => 'DATA_DIR'
   ,bad_file        => 'CLOBrows.bad'
   );
END;
/

-- Use ALTER TABLE to target the appropriate file(s)
alter table "stage_table_source" location ('demo_datafile.nt');
```

```
-- Load the staging table (note: table names are case sensitive)
BEGIN
  sem_apis.load_into_staging_table(
    staging_table => 'STAGE_TABLE'
   ,source_table  => 'stage_table_source'
   ,input_format  => 'N-QUAD');
END;
/
```

Rows where the objects and graph URIs (combined) are longer than `NETWORK_MAX_STRING_SIZE` bytes will be rejected and stored in a "bad" file. However, you can load these rejected rows by inserting them into the application table using SQL INSERT statements (see Loading RDF Data Using INSERT Statements).

Example 1-109 shows the use of an external table to load a staging table.

1.11.1.2 Recording Event Traces During Bulk Loading

If a table named RDF$ET_TAB exists in the invoker's schema and if the network owner user has been granted the INSERT and UPDATE privileges on this table, event traces for some of the tasks performed during executions of the SEM_APIS.BULK_LOAD_RDF_GRAPH procedure will be added to the table. You may find the content of this table useful if you ever need to report any problems in bulk load. The RDF$ET_TAB table must be created as follows:

```
CREATE TABLE RDF$ET_TAB (
  proc_sid VARCHAR2(128),
  proc_sig VARCHAR2(200),
  event_name varchar2(200),
  start_time timestamp,
  end_time timestamp,
  start_comment varchar2(1000) DEFAULT NULL,
  end_comment varchar2(1000) DEFAULT NULL
);
-- Grant privileges on RDF$ET_TAB to network owner if network owner
-- is not the owner of RDF$ET_TAB
GRANT SELECT, INSERT, UPDATE on RDF$ET_TAB to <network_owner>;
```

1.11.2 Loading RDF Data Using INSERT Statements

To load RDF data using INSERT statements, the data should be encoded using `< >` (angle brackets) for URIs, `_:` (underscore colon) for blank nodes, and `" "` (quotation marks) for literals. Spaces are not allowed in URIs or blank nodes. Use the SDO_RDF_TRIPLE_S constructor to insert the data, as described in Constructors for Inserting Triples. You must have INSERT privilege on the application table.

> **Note:**
>
> If URIs are not encoded with `< >` and literals with `" "`, statements will still be processed. However, the statements will take longer to load, since they will have to be further processed to determine their VALUE_TYPE values.

The following example assumes an RDF network named NET1 owned by RDFUSER. It includes statements with URIs, a blank node, a literal, a literal with a language tag, and a typed literal:

```
INSERT INTO nsu_data VALUES (SDO_RDF_TRIPLE_S('nsu','<http://nature.example.com/nsu/rss.rdf>',
  '<http://purl.org/rss/1.0/title>', '"Nature''s Science Update"', 'RDFUSER', 'NET1'));
INSERT INTO nsu_data VALUES (SDO_RDF_TRIPLE_S('nsu', '_:BNSEQN1001A',
  '<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>',
  '<http://www.w3.org/1999/02/22-rdf-syntax-ns#Seq>', 'RDFUSER', 'NET1'));
INSERT INTO nsu_data VALUES (SDO_RDF_TRIPLE_S('nsu',
  '<http://nature.example.com/cgi-taf/dynapage.taf?file=/nature/journal/v428/n6978/index.html>',
  '<http://purl.org/dc/elements/1.1/language>', '"English"@en-GB', 'RDFUSER', 'NET1'));
INSERT INTO nature VALUES (SDO_RDF_TRIPLE_S('nsu', '<http://dx.doi.org/10.1038/428004b>',
  '<http://purl.org/dc/elements/1.1/date>', '"2004-03-04"^^xsd:date', 'RDFUSER', 'NET1'));
```

- Loading Data into Named Graphs Using INSERT Statements

1.11.2.1 Loading Data into Named Graphs Using INSERT Statements

To load an RDF triple with a non-null graph name using an INSERT statement, you must append the graph name, enclosed within angle brackets (< >), after the RDF graph name and colon (:) separator character, as shown in the following example:

```
INSERT INTO articles_rdf_data VALUES (
  SDO_RDF_TRIPLE_S ('articles:<http://examples.com/ns#Graph1>',
    '<http://nature.example.com/Article101>',
    '<http://purl.org/dc/elements/1.1/creator>',
    '"John Smith"', 'RDFUSER', 'NET1'));
```

1.11.3 Exporting RDF Data

This section contains the following topics related to exporting RDF data, that is, retrieving RDF data from Oracle Database where the results are in N-Triple or N-Quad format that can be stored in a staging table.

- Retrieving RDF Data from an Application Table
- Retrieving RDF Data from an RDF Graph
- Removing RDF Graph Information from Retrieved Blank Node Identifiers

1.11.3.1 Retrieving RDF Data from an Application Table

RDF data can be retrieved from an application table using the member functions of SDO_RDF_TRIPLE_S, as shown in Example 1-110 (where the output is reformatted for readability). The example assumes a RDF network named NET1 owned by a database user named RDFUSER.

Example 1-110 Retrieving RDF Data from an Application Table

```
--
-- Retrieves model-graph, subject, predicate, and object
--
SQL> SELECT a.triple.GET_MODEL('RDFUSER','NET1') AS model_graph,
            a.triple.GET_SUBJECT('RDFUSER','NET1') AS sub,
            a.triple.GET_PROPERTY('RDFUSER','NET1') pred,
            a.triple.GET_OBJ_VALUE('RDFUSER','NET1') obj
     FROM RDFUSER.NET1#RDFT_ARTICLES a;

MODEL_GRAPH
--------------------------------------------------------------------------------
SUB
--------------------------------------------------------------------------------
PRED
```

```
--------------------------------------------------------------------------------
OBJ
--------------------------------------------------------------------------------
ARTICLES
<http://nature.example.com/Article1>
<http://purl.org/dc/elements/1.1/title>
"All about XYZ"


ARTICLES
<http://nature.example.com/Article1>
<http://purl.org/dc/elements/1.1/creator>
"Jane Smith"


ARTICLES
<http://nature.example.com/Article1>
<http://purl.org/dc/terms/references>
<http://nature.example.com/Article2>


ARTICLES
<http://nature.example.com/Article1>
<http://purl.org/dc/terms/references>
<http://nature.example.com/Article3>


ARTICLES
<http://nature.example.com/Article2>
<http://purl.org/dc/elements/1.1/title>
"A review of ABC"


ARTICLES
<http://nature.example.com/Article2>
<http://purl.org/dc/elements/1.1/creator>
"Joe Bloggs"


ARTICLES
<http://nature.example.com/Article2>
<http://purl.org/dc/terms/references>
<http://nature.example.com/Article3>


7 rows selected.
```

1.11.3.2 Retrieving RDF Data from an RDF Graph

RDF data can be retrieved from an RDF graph using the SEM_MATCH table function (described in Using the SEM_MATCH Table Function to Query RDF Data), as shown in Example 1-111. The example assumes an RDF network named NET1 owned by a database user named RDFUSER.

Example 1-111 Retrieving RDF Data from an RDF Graph

```
--
-- Retrieves graph, subject, predicate, and object
--
SQL> select to_char(g$rdfterm) graph, to_char(x$rdfterm) sub, to_char(p$rdfterm) pred,
y$rdfterm obj from table(sem_match('Select ?g ?x ?p ?y  WHERE { { GRAPH ?g {?x ?p ?y} }
UNION { ?x ?p ?y }}',sem_models('articles'),null,null,null,null,' STRICT_DEFAULT=T
```

```
PLUS_RDFT=T ',null,null,'RDFUSER','NET1'));

GRAPH
------------------------------------------------------------
SUB
------------------------------------------------------------
PRED
------------------------------------------------------------
OBJ
---------------------------------------------------------------------------
<http://examples.com/ns#Graph1>
_:m99g3C687474703A2F2F6578616D706C65732E636F6D2F6E73234772617068313Egmb2
<http://purl.org/dc/elements/1.1/creator>
_:m99g3C687474703A2F2F6578616D706C65732E636F6D2F6E73234772617068313Egmb1

<http://examples.com/ns#Graph1>
<http://nature.example.com/Article102>
<http://purl.org/dc/elements/1.1/creator>
_:m99g3C687474703A2F2F6578616D706C65732E636F6D2F6E73234772617068313Egmb1

<http://examples.com/ns#Graph1>
<http://nature.example.com/Article101>
<http://purl.org/dc/elements/1.1/creator>
"John Smith"

<http://nature.example.com/Article1>
<http://purl.org/dc/elements/1.1/creator>
"Jane Smith"
```

1.11.3.3 Removing RDF Graph Information from Retrieved Blank Node Identifiers

Blank node identifiers retrieved during the retrieval of RDF data can be trimmed to remove the occurrence of RDF graph information using the transformations shown in the code excerpt in Example 1-112, which are applicable to VARCHAR2 (for example, subject component) and CLOB (for example, object component) data, respectively.

Example 1-113 shows the results obtained after using these two transformations in Example 1-112 on the `sub` and `obj` columns, respectively, using the RDF data retrieval query described in Retrieving RDF Data from an RDF Graph.

Example 1-112 Retrieving RDF Data from an Application Table

```
--
-- Transformation on column "sub VARCHAR2"
-- holding blank node identifier values
--
Select (case substr(sub,1,2) when '_:' then '_:' || substr(sub,instr(sub,'m',1,2)+1)
else sub end) from …
--
-- Transformation on column "obj CLOB"
-- holding blank node identifier values
--
Select (case dbms_lob.substr(obj,2,1) when '_:' then to_clob('_:' ||
substr(obj,instr(obj,'m',1,2)+1)) else obj end) from …
```

Example 1-113 Results from Applying Transformations from Example 1-112

```
--
-- Results obtained by applying transformations on the sub and pred cols
--
SQL> select (case substr(sub,1,2) when '_:' then '_:' ||
substr(sub,instr(sub,'m',1,2)+1) else sub end) sub, pred, (case dbms_lob.substr(obj,2,1)
```

```
when '_:' then to_clob('_:' || substr(obj,instr(obj,'m',1,2)+1)) else obj end) obj from
(select to_char(g$rdfterm) graph, to_char(x$rdfterm) sub, to_char(p$rdfterm) pred,
y$rdfterm obj from table(sem_match('Select ?g ?x ?p ?y  WHERE { { GRAPH ?g {?x ?p ?y} }
UNION { ?x ?p ?y }}',sem_models('articles'),null,null,null,null,' STRICT_DEFAULT=T
PLUS_RDFT=T ',null,null,'RDFUSER','NET1'));


SUB
------------------------------------------------------------
PRED
------------------------------------------------------------
OBJ
---------------------------------------------------------------------------
_:b2
<http://purl.org/dc/elements/1.1/creator>
_:b1

<http://nature.example.com/Article102>
<http://purl.org/dc/elements/1.1/creator>
_:b1
```

1.11.4 Exporting or Importing an RDF Network Using Oracle Data Pump

Effective with Oracle Database Release 12.1, you can export and import an RDF network using the full database export and import features of the Oracle Data Pump utility. The network is moved as part of the full database export or import, where the whole database is represented in an Oracle dump (`.dmp`) file.

The following usage notes apply to using Data Pump to export or import a RDF network:

- The target database for an import must have the RDF Graph software installed, and there cannot be a pre-existing RDF network.
- RDF networks using fine-grained access control (triple-level or resource-level OLS or VPD) cannot be exported or imported.
- RDF document indexes for SEM_CONTAINS (MDSYS.SEMCONTEXT index type) and semantic indexes for SEM_RELATED (MDSYS.SEM_INDEXTYPE index type) must be dropped before an export and re-created after an import.
- Only default privileges for RDF network objects (those that exist just after object creation) are preserved during export and import. For example, if user A creates an RDF graph `M` and grants SELECT on RDFM_M to user B, only user A's SELECT privilege on RDFM_M will be present after the import. User B will not have SELECT privilege on RDFM_M after the import. Instead, user B's SELECT privilege will have to be granted again.
- The Data Pump command line option `transform=oid:n` must be used when exporting or importing RDF network data. For example, use a command in the following format:

  ```
  impdp system/<password-for-system> directory=dpump_dir dumpfile=rdf.dmp full=YES
  version=12 transform=oid:n
  ```

For Data Pump usage information and examples, see the relevant chapters in Part I of *Oracle Database Utilities*.

1.11.5 Moving, Restoring, and Appending an RDF Network

The SEM_APIS package includes utility procedures for transferring data into and out of an RDF network.

The contents of an RDF network can be moved to a staging schema. an RDF network in a staging schema can then be (1) exported with Oracle Data Pump or a similar tool, (2) appended to a different RDF network, or (3) restored back into the source RDF network. Move, restore and append operations mostly use partition exchange to move data rather than SQL inserts to copy data. Consequently, these operations are very efficient.

The procedures to move, restore, and append RDF network data are:

- SEM_APIS.MOVE_RDF_NETWORK_DATA
- SEM_APIS.RESTORE_RDF_NETWORK_DATA
- SEM_APIS.APPEND_RDF_NETWORK_DATA

Special Considerations When Performing Move, Restore, and Append Operations

Move, restore, and append operations are not supported for RDF networks that use any of the following features:

- Domain indexes on the RDF_VALUE$ table (for example, spatial indexes)
- Oracle Label Security for RDF
- Semantic indexing for documents
- Incremental inference

Domain indexes and inferred graphs that use incremental inference should be dropped before moving the RDF network and then recreated after any subsequent restore or append operations.

Some restrictions apply to the target network used for an append operation.

- The set of RDF terms in the target network must be a subset of the set of RDF terms in the source network.
- The set of model (referred to as RDF graph) IDs used in the source and target RDF networks must be disjoint.
- The set of entailment (referred to as inferred graph) IDs used in the source and target RDF networks must be disjoint.
- The set of rulebase IDs used in the source and target RDF networks must be disjoint, with the exception of built-in rulebases such as OWL2RL.

Example 1-114 Moving and Exporting a Schema Private RDF Network

This first example uses Data Pump Export to export relevant network data to multiple `.dmp` files, so that the data can be imported into an RDF network in another database (as shown in the second example).

This example performs the following major actions.

1. Creates a directory for a Data Pump Export operation.
2. Creates a database user (RDFEXPIMPU) that will hold the output of the export of the RDF network.
3. Moves the RDF network data to the RDFEXPIMPU schema.
4. Uses Data Pump to export the moved RDF network data.
5. Uses Data Pump to export any user tables referenced in RDF views.
6. Optionally, restores the RDF network data in the current network.

Note that the example assumes that the schema-private network is named as `NET1` and it is owned by `RDFUSER`. It also assumes that the tables `EMP`, `WORKED_FOR`, and `DEPT`, owned by `RDFUSER`, are used in the RDF view RDF graph(s) in the network.

```
conn sys/<password_for_sys> as sysdba;

-- create directory for datapump export
create directory dpump_dir as '<path_to_directory>';
grant read,write on directory dpump_dir to public;

-- create user to hold exported RDF network
grant connect, resource, unlimited tablespace to rdfexpimpu identified by
<password_for_rdfexpimpu>;

-- connect as a privileged user to move the network
conn system/<password_for_system>
-- move RDF network data to RDFEXPIMPU schema
exec sem_apis.move_rdf_network_data(dest_schema=>'RDFEXPIMPU',
network_owner=>'RDFUSER', network_name=>'NET1');

-- export moved network data with datapump
-- export rdfexpimpu schema
host expdp rdfexpimpu/<password_for_rdfexpimpu> DIRECTORY=dpump_dir
DUMPFILE=expuser.dmp version=12.2 logfile=export_move_sem_network_data.log

-- export any user tables referenced in RDF Views
host expdp rdfuser/<password_for_rdfuser> tables=EMP,WORKED_FOR,DEPT
DIRECTORY=dpump_dir DUMPFILE=exp_rdfviewtabs.dmp version=12.2
logfile=export_move_rdfview_tabs.log

-- optionally restore the network data or drop the source RDF network
exec sem_apis.restore_rdf_network_data(from_schema=>'RDFEXPIMPU',
network_owner=>'RDFUSER', network_name=>'NET1');
```

Example 1-115 Importing and Appending a Schema Private RDF Network

This second example uses Data Pump Import to import relevant network data (from the first example), creates necessary database users, creates a new MDSYS-owned RDF network, and "appends" the imported network data into the newly created network.

This example performs the following major actions.

1. Creates a database user (RDFEXPIMPU), if it does not already exist in the database, that will hold the output of the export of the RDF network.
2. Uses Data Pump to import any RDF view component tables and previously moved RDF network data.
3. Creates a new RDF network in which the imported data is to be appended.
4. Appends the imported data into the newly created RDF network.

```
conn sys/<password_for_sys>

-- create a user to hold the imported RDF network data
grant connect, resource, unlimited tablespace to rdfexpimpu identified by
<password_for_rdfexpimpu>;
```

```
-- create the network owner
grant connect, resource, unlimited tablespace to rdfuser identified by
<password_for_rdfuser>;

conn system/<password_for_system>

-- import any RDF view component tables
host impdp rdfuser/<password_for_rdfuser> tables=EMP,WORKED_FOR,DEPT
DIRECTORY=dpump_dir DUMPFILE=exp_rdfviewtabs.dmp version=12.2
logfile=import_append_rdfview_tabs.log

-- import the previously moved RDF network
host impdp rdfexpimpu/<password_for_rdfexpimpu> DIRECTORY=dpump_dir
DUMPFILE=expuser.dmp version=12.2 logfile=import_append_atabs.log

-- create a new RDF network in which to append the imported one
exec sem_apis.create_rdf_network('rdf_tablespace', network_owner=>'RDFUSER',
network_name=>'NET1');

-- append the imported RDF network
exec sem_apis.append_rdf_network_data(from_schema=>'RDFEXPIMPU',
network_owner=>'RDFUSER', network_name=>'NET1');
```

1.11.6 Purging Unused Values

Deletion of triples over time may lead to a subset of the values in the RDF_VALUE$ table becoming unused in any of the RDF triples or rules currently in the RDF network. If the count of such unused values becomes large and a significant portion of the RDF_VALUE$ table, you may want to purge the unused values using the SEM_APIS.PURGE_UNUSED_VALUES subprogram.

Event traces for tasks performed during the purge operation may be recorded into the RDF$ET_TAB table, if present in the invoker's schema, as described in Recording Event Traces During Bulk Loading.

1.12 Using RDF Network Indexes

RDF network indexes are nonunique B-tree indexes that you can add, alter, and drop for use with RDF graphs and inferred graphs in a RDF network.

You can use such indexes to tune the performance of SEM_MATCH queries on the RDF graphs and inferred graphs in the network. As with any indexes, RDF network indexes enable index-based access that suits your query workload. This can lead to substantial performance benefits, such as in the following example scenarios:

- If your graph pattern is `'{<John> ?p <Mary>}'`, you may want to have a usable `'CSPGM'`or `'SCPGM'` index for the target RDF graphs and on the corresponding inferred graph, if used in the query.
- If your graph pattern is `'{?x <talksTo> ?y . ?z ?p ?y}'`, you may want to have a usable RDF network index on the relevant RDF graphs and inferred graph, with `C` as the leading key (for example, `'CPSGM'`).

However, using RDF network indexes can affect overall performance by increasing the time required for DML, load, and inference operations.

You can create and manage RDF network indexes using the following subprograms:

- SEM_APIS.ADD_NETWORK_INDEX
- SEM_APIS.ALTER_INDEX_ON_RDF_GRAPH
- SEM_APIS.ALTER_INDEX_ON_INFERRED_GRAPH
- SEM_APIS.DROP_NETWORK_INDEX

All of these subprograms have an `index_code` parameter, which can contain any sequence of the following letters (without repetition): `P`, `C`, `S`, `G`, `M`. These letters used in the index_code correspond to the following columns in the SEMM_* and SEMI_* views: P_VALUE_ID, CANON_END_NODE_ID, START_NODE_ID, G_ID, and MODEL_ID.

The SEM_APIS.ADD_NETWORK_INDEX procedure creates an RDF network index that results in the creation of a nonunique B-tree index in UNUSABLE status for each of the existing RDF graphs and inferred graphs. The name of the index is RDF_LNK_<*index_code*>_IDX and the index is owned by the network owner. This operation is allowed only if the invoker has DBA role or is the network owner. The following example shows creation of the `PSCGM` index with the following key: <P_VALUE_ID, START_NODE_ID, CANON_END_NODE_ID, G_ID, MODEL_ID>.

```
EXECUTE SEM_APIS.ADD_NETWORK_INDEX('PSCGM' network_owner=>'RDFUSER',
network_name=>'NET1');
```

After you create a RDF network index, each of the corresponding nonunique B-tree indexes is in the UNUSABLE status, because making it usable can cause significant time and resources to be used, and because subsequent index maintenance operations might involve performance costs that you do not want to incur. You can make a RDF network index usable or unusable for specific RDF graphs or inferred graphs that you own by calling the SEM_APIS.ALTER_INDEX_ON_RDF_GRAPH and SEM_APIS.ALTER_INDEX_ON_INFERRED_GRAPH procedures and specifying `'REBUILD'` or `'UNUSABLE'` as the `command` parameter. Thus, you can experiment by making different RDF network indexes usable and unusable, and checking for any differences in performance. For example, the following statement makes the `PSCGM` index usable for the `FAMILY` RDF graph:

```
EXECUTE SEM_APIS.ALTER_INDEX_ON_RDF_GRAPH('FAMILY','PSCGM','REBUILD'
network_owner=>'RDFUSER', network_name=>'NET1');
```

Also note the following:

- Independent of any RDF network indexes that you create, when an RDF network is created, one of the indexes that is automatically created is an index that you can manage by referring to the `index_code` as `'PSCGM'` when you call the subprograms mentioned in this section.
- When you create a new RDF graph or a new inferred graph, a new nonunique B-tree index is created for each of the RDF network indexes, and each such B-tree index is in the USABLE status.
- Including the MODEL_ID column in an RDF network index key (by including 'M' in the `index_code` value) may improve query performance. This is particularly relevant when RDF graph collections are used.
- SEM_NETWORK_INDEX_INFO View

1.12.1 SEM_NETWORK_INDEX_INFO View

Information about all network indexes on RDF graphs and inferred graphs is maintained in the SEM_NETWORK_INDEX_INFO view, which includes (a partial list) the columns shown in Table 1-29 and one row for each network index.

Table 1-29 SEM_NETWORK_INDEX_INFO View Columns (Partial List)

Column Name	Data Type	Description
NAME	VARCHAR2(30)	Name of the RDF graph or inferred graph
TYPE	VARCHAR2(10)	Type of object on which the index is built: `MODEL`, `ENTAILMENT`, or `NETWORK`
ID	NUMBER	ID number for the RDF graph or inferred graph, or zero (0) for an index on the network
INDEX_CODE	VARCHAR2(25)	Code for the index (for example, `PSCGM`).
INDEX_NAME	VARCHAR2(30)	Name of the index (for example, `RDF_LNK_PSCGM_IDX`)
LAST_REFRESH	TIMESTAMP(6) WITH TIME ZONE	Timestamp for the last time this content was refreshed

In addition to the columns listed in Table 1-29, the SEM_NETWORK_INDEX_INFO view contains columns from the ALL_INDEXES and ALL_IND_PARTITIONS views (both described in *Oracle Database Reference*), including:

- From the ALL_INDEXES view: UNIQUENESS, COMPRESSION, PREFIX_LENGTH
- From the ALL_IND_PARTITIONS view: STATUS, TABLESPACE_NAME, BLEVEL, LEAF_BLOCKS, NUM_ROWS, DISTINCT_KEYS, AVG_LEAF_BLOCKS_PER_KEY, AVG_DATA_BLOCKS_PER_KEY, CLUSTERING_FACTOR, SAMPLE_SIZE, LAST_ANALYZED

Note that the information in the SEM_NETWORK_INDEX_INFO view may sometimes be stale. You can refresh this information by using the SEM_APIS.REFRESH_NETWORK_INDEX_INFO procedure.

1.13 Using Data Type Indexes

Data type indexes are indexes on the values of typed literals stored in an RDF network.

These indexes may significantly improve the performance of SEM_MATCH queries involving certain types of FILTER expressions. For example, a data type index on `xsd:dateTime` literals may speed up evaluation of the filter `(?x < "1929-11-16T13:45:00Z"^^xsd:dateTime)`. Indexes can be created for several data types, which are listed in Table 1-30.

Table 1-30 Data Types for Data Type Indexing

Data Type URI	Oracle Type	Index Type
http://www.w3.org/2001/XMLSchema#decimal	NUMBER	Non-unique B-tree (creates a single index for all xsd numeric types, including `xsd:float`, `xsd:double`, and `xsd:decimal` and all of its subtypes)

Table 1-30 (Cont.) Data Types for Data Type Indexing

Data Type URI	Oracle Type	Index Type
http://www.w3.org/2001/XMLSchema#string	VARCHAR2	Non-unique B-tree (creates a single index for `xsd:string` typed literals and plain literals)
http://www.w3.org/2001/XMLSchema#time	TIMESTAMP WITH TIMEZONE	Non-unique B-tree
http://www.w3.org/2001/XMLSchema#date	TIMESTAMP WITH TIMEZONE	Non-unique B-tree
http://www.w3.org/2001/XMLSchema#dateTime	TIMESTAMP WITH TIMEZONE	Non-unique B-tree
http://xmlns.oracle.com/rdf/text	(Not applicable)	CTXSYS.CONTEXT
http://xmlns.oracle.com/rdf/geo/WKTLiteral	SDO_GEOMETRY	SPATIAL_INDEX
http://www.opengis.net/geosparql#wktLiteral	SDO_GEOMETRY	SPATIAL_INDEX
http://www.opengis.net/geosparql#gmlLiteral	SDO_GEOMETRY	SPATIAL_INDEX
http://xmlns.oracle.com/rdf/like	VARCHAR2	Non-unique B-tree

The suitability of data type indexes depends on your query workload. Data type indexes on `xsd` data types can be used for filters that compare a variable with a constant value, and are particularly useful when queries have an unselective graph pattern with a very selective filter condition. Appropriate data type indexes are required for queries with spatial or text filters.

While data type indexes improve query performance, overhead from incremental index maintenance can degrade the performance of DML and bulk load operations on the RDF network. For bulk load operations, it may often be faster to drop data type indexes, perform the bulk load, and then re-create the data type indexes. As it is time consuming to create a text index on large amounts of text data, `nologging` is enabled by default when the text index is created. The logging can be enabled by specifying '`LOGGING=T`' in the options field of `add_datatype_index` API for the text index.

You can add, alter, and drop data type indexes using the following procedures, which are described in SEM_APIS Package Subprograms:

- SEM_APIS.ADD_DATATYPE_INDEX
- SEM_APIS.ALTER_DATATYPE_INDEX
- SEM_APIS.DROP_DATATYPE_INDEX

Information about existing data type indexes is maintained in the SEM_DTYPE_INDEX_INFO view, which has the columns shown in Table 1-31 and one row for each data type index.

Table 1-31 SEM_DTYPE_INDEX_INFO View Columns

Column Name	Data Type	Description
DATATYPE	VARCHAR2(51)	Data type URI
INDEX_NAME	VARCHAR2(30)	Name of the index
STATUS	VARCHAR2(8)	Status of the index: `USABLE` or `UNUSABLE`

Table 1-31 (Cont.) SEM_DTYPE_INDEX_INFO View Columns

Column Name	Data Type	Description
TABLESPACE_NAME	VARCHAR2(30)	Tablespace for the index
FUNCIDX_STATUS	VARCHAR2(8)	Status of the function-based index: `NULL`, `ENABLED`, or `DISABLED`

You can use the `HINT0` hint to ensure that data type indexes are used during query evaluation, as shown in Example 1-116, which finds all grandfathers who were born before November 16, 1929.

Example 1-116 Using HINT0 to Ensure Use of Data Type Index

```
SELECT x, y
  FROM TABLE(SEM_MATCH(
    'PREFIX : <http://www.example.org/family/>
     SELECT ?x ?y
     WHERE {?x :grandParentOf ?y . ?x rdf:type :Male . ?x :birthDate ?bd
       FILTER (?bd <= "1929-11-15T23:59:59Z"^^xsd:dateTime) }',
    SEM_Models('family'),
    SEM_Rulebases('RDFS','family_rb'),

    null, null, null,
    'HINT0={ LEADING(?bd) INDEX(?bd rdf_v$dateTime_idx) }
             FAST_DATE_FILTER=T',
    null, null,
    'RDFUSER', 'NET1' ));
```

1.14 Managing Statistics for the RDF Graphs and RDF Network

Statistics are critical to the performance of SPARQL queries and OWL inference against RDF data stored in an Oracle database.

Oracle Database Release 11*g* introduced SEM_APIS.ANALYZE_RDF_GRAPH, SEM_APIS.ANALYZE_INFERRED_GRAPH, and SEM_PERF.GATHER_STATS to analyze RDF data and keep statistics up to date. These APIs are straightforward to use and they are targeted at regular users who may not care about the internal details about table and partition statistics.

You can export, import, set, and delete the RDF graph and inferred graph statistics, and can export, import, and delete network statistics, using the following subprograms:

- SEM_APIS.DELETE_ENTAILMENT_STATS
- SEM_APIS.DELETE_MODEL_STATS
- SEM_APIS.EXPORT_ENTAILMENT_STATS
- SEM_APIS.EXPORT_MODEL_STATS
- SEM_APIS.IMPORT_ENTAILMENT_STATS
- SEM_APIS.IMPORT_MODEL_STATS
- SEM_APIS.SET_ENTAILMENT_STATS
- SEM_APIS.SET_MODEL_STATS
- SEM_PERF.DELETE_NETWORK_STATS

- SEM_PERF.DROP_EXTENDED_STATS
- SEM_PERF.EXPORT_NETWORK_STATS
- SEM_PERF.IMPORT_NETWORK_STATS

This section contains the following topics related to managing statistics for RDF graphs and the RDF network.

- Saving Statistics at the RDF Graph Level
- Restoring Statistics at the RDF Graph Level
- Saving Statistics at the Network Level
- Dropping Extended Statistics at the Network Level
- Restoring Statistics at the Network Level
- Setting Statistics at the RDF Graph Level
- Deleting Statistics at the RDF Graph Level

1.14.1 Saving Statistics at the RDF Graph Level

If queries and inference against an existing RDF graph are executed efficiently, as the owner of the RDF graph, you can save the statistics of the existing RDF graph.

```
-- Login as the RDF graph owner (for example, SCOTT)
-- Create a stats table. This is required.
execute dbms_stats.create_stat_table('scott','rdf_stat_tab');

-- Now export the statistics of RDF graph TEST
execute sem_apis.export_model_stats('TEST','rdf_stat_tab', 'model_stat_saved_on_AUG_10',
true, 'SCOTT', 'OBJECT_STATS', network_owner=>'RDFUSER', network_name=>'NET1');
```

You can also save the statistics of an inferred graph by using SEM_APIS.EXPORT_ENTAILMENT_STATS .

```
execute
sem_apis.create_inferred_graph('test_inf',sem_models('test'),sem_rulebases('owl2rl'),0,nu
ll,network_owner=>'RDFUSER',network_name=>'NET1');
PL/SQL procedure successfully completed.

execute sem_apis.export_entailment_stats('TEST_INF','rdf_stat_tab',
'inf_stat_saved_on_AUG_10', true, 'SCOTT', 'OBJECT_STATS', network_owner=>'RDFUSER',
network_name=>'NET1');
```

1.14.2 Restoring Statistics at the RDF Graph Level

As the owner of an RDF graph, can restore the statistics that were previously saved with SEM_APIS.EXPORT_MODEL_STATS . This may be necessary if updates have been applied to this RDF graph and statistics have been re-collected. A change in statistics might cause a plan change to existing SPARQL queries, and if such a plan change is undesirable, then an old set of statistics can be restored.

```
execute sem_apis.import_model_stats('TEST','rdf_stat_tab', 'model_stat_saved_on_AUG_10',
true, 'SCOTT', false, true, 'OBJECT_STATS', network_owner=>'RDFUSER',
network_name=>'NET1');
```

You can also restore the statistics of an inferred graph by using SEM_APIS.IMPORT_ENTAILMENT_STATS .

```
execute sem_apis.import_entailment_stats('TEST','rdf_stat_tab',
'inf_stat_saved_on_AUG_10', true, 'SCOTT', false, true, 'OBJECT_STATS',
network_owner=>'RDFUSER', network_name=>'NET1');
```

1.14.3 Saving Statistics at the Network Level

You can save statistics at the network level.

```
-- Network owners and DBAs have privileges to gather network-wide
-- statistics with the SEM_PERF package.
--
-- This example assumes a schema-private RDF network named NET1
-- owned by RDFUSER.
--

conn RDFUSER/<password>

execute dbms_stats.create_stat_table('RDFUSER','rdf_stat_tab');

--
-- This API call will save the statistics of both the RDF_VALUE$ table
-- and RDF_LINK$ table
--
execute sem_perf.export_network_stats('rdf_stat_tab', 'NETWORK_ALL_saved_on_Aug_10',
true, 'RDFUSER', 'OBJECT_STATS', network_owner=>'RDFUSER', network_name=>'NET1');

--
-- Alternatively, you can save statistics of only the RDF_VALUE$ table
--
execute sem_perf.export_network_stats('rdf_stat_tab',
'NETWORK_VALUE_TAB_saved_on_Aug_10', true, 'RDFUSER', 'OBJECT_STATS', options=>
mdsys.sdo_rdf.VALUE_TAB_ONLY, network_owner=>'RDFUSER', network_name=>'NET1');

--
-- Or, you can save statistics of only the RDF_LINK$ table
--
execute sem_perf.export_network_stats('rdf_stat_tab',
'NETWORK_LINK_TAB_saved_on_Aug_10', true, 'RDFUSER', 'OBJECT_STATS', options=>
mdsys.sdo_rdf.LINK_TAB_ONLY, network_owner=>'RDFUSER', network_name=>'NET1');
```

1.14.4 Dropping Extended Statistics at the Network Level

By default, SEM_PERF.GATHER_STATS creates extended statistics with column groups on the RDF_LINK$ table. The privileged user from Saving Statistics at the Network Level can drop these column groups using SEM_PERF.DROP_EXTENDED_STATS.

```
connect RDFUSER/<password>
execute sem_perf.drop_extended_stats(network_owner=>'RDFUSER', network_name=>'NET1');
```

See also the information about managing extended statistics in *Oracle Database SQL Tuning Guide*.

1.14.5 Restoring Statistics at the Network Level

The privileged user from Saving Statistics at the Network Level can restore the network level statistics using SEM_PERF.IMPORT_NETWORK_STATS .

```
conn RDFUSER/<password>

execute sem_perf.import_network_stats('rdf_stat_tab', 'NETWORK_ALL_saved_on_Aug_10',
```

```
true, 'RDFUSER', false, true, 'OBJECT_STATS', network_owner=>'RDFUSER',
network_name=>'NET1');
```

1.14.6 Setting Statistics at the RDF Graph Level

As the owner of an RDF graph, you can manually adjust the statistics for this RDF graph. (However, before you adjust statistics, you should save the statistics first so that they can be restored if necessary.) The following example sets two metrics: number of rows and number of blocks for the RDF graph.

```
execute sem_apis.set_model_stats('TEST', numrows=>10,
numblks=>1,no_invalidate=>false,network_owner=>'RDFUSER',network_name=>'NET1');
```

You can also set the statistics for the inferred graph by using SEM_APIS.SET_ENTAILMENT_STATS .

```
execute sem_apis.set_entailment_stats('TEST_INF', numrows=>10,
numblks=>1,no_invalidate=>false,network_owner=>'RDFUSER',network_name=>'NET1');
```

1.14.7 Deleting Statistics at the RDF Graph Level

Removing statistics can also have an impact on execution plans. As owner of an RDF graph, you can remove the statistics for the graph.

```
execute sem_apis.delete_model_stats('TEST', no_invalidate=> false,
network_owner=>'RDFUSER', network_name=>'NET1');
```

You can also remove the statistics for the inferred graph by using SEM_APIS.DELETE_ENTAILMENT_STATS. (However, before you remove statistics of an RDF graph or an inferred graph, you should save the statistics first so that they can be restored if necessary.)

```
execute sem_apis.delete_entailment_stats('TEST_INF', no_invalidate=> false,
network_owner=>'RDFUSER', network_name=>'NET1');
```

1.15 Support for SPARQL Update Operations on an RDF Graph

Effective with Oracle Database Release 12.2, you can perform SPARQL Update operations on an RDF graph.

> **Note:**
>
> SPARQL update operations on an RDF graph is supported only if Oracle JVM is enabled on your Oracle Autonomous Database Serverless deployments. To enable Oracle JVM, see Use Oracle Java in *Using Oracle Autonomous Database Serverless* for more information.

The W3C provides SPARQL 1.1 Update (https://www.w3.org/TR/2013/REC-sparql11-update-20130321/), an update language for RDF graphs. SPARQL 1.1 Update is supported in Oracle Database semantic technologies through the SEM_APIS.UPDATE_RDF_GRAPH procedure.

Before performing any SPARQL Update operations on an RDF graph, some prerequisites apply:

- The SEM_APIS.CREATE_SPARQL_UPDATE_TABLES procedure should be run in the schema of each user that will be using the SEM_APIS.UPDATE_RDF_GRAPH procedure.
- To update an RDF graph, the user should have `SELECT`, `INSERT`, `DELETE`, `UPDATE`, and `QUERY` privileges on the network and the target RDF graph. Note that these privileges are automatically present for the network owner. See Sharing Schema-Private RDF Networks to enable other users to update the RDF graph.
- To run a LOAD operation, the user must have the CREATE ANY DIRECTORY and DROP ANY DIRECTORY privileges, or the user must be granted READ privileges on an existing directory object whose name is supplied in the `options` parameter.

The following examples show update operations being performed on an RDF graph. These examples assume a schema-private RDF network named NET1 owned by a database user named RDFUSER.

Example 1-117 INSERT DATA Operation

This example shows an INSERT DATA operation that inserts several triples in the default `electronics` graph.

```
-- Dataset before operation:
#Empty default graph
-- Update operation:
BEGIN
  sem_apis.update_rdf_graph('electronics',
   'PREFIX : <http://www.example.org/electronics/>
    INSERT DATA {
       :camera1 :name "Camera 1" .
       :camera1 :price 120 .
       :camera1 :cameraType :Camera .
       :camera2 :name "Camera 2" .
       :camera2 :price 150 .
       :camera2 :cameraType :Camera .
      } ',
   network_owner=>'RDFUSER', network_name=>'NET1');
END;
/

-- Dataset after operation:
@prefix : <http://www.example.org/electronics/>
#Default graph
:camera1 :name "Camera 1";
         :price 120;
         :cameraType :Camera .
:camera2 :name "Camera 2";
         :price 150;
         :cameraType :Camera .
```

Example 1-118 DELETE DATA Operation

This example shows a DELETE DATA operation that removes a single triple from the default `electronics` RDF graph.

```
-- Dataset before operation:
@prefix : <http://www.example.org/electronics/>
#Default graph
```

```
:camera1 :name "Camera 1";
         :price 120;
         :cameraType :Camera .
:camera2 :name "Camera 2";
         :price 150;
         :cameraType :Camera .
-- Update operation:
BEGIN
  sem_apis.update_rdf_graph('electronics',
   'PREFIX : <http://www.example.org/electronics/>
    DELETE DATA { :camera1 :price 120 . } ',
   network_owner=>'RDFUSER', network_name=>'NET1');
END;
/

-- Dataset after operation:
@prefix : <http://www.example.org/electronics/>
#Default graph
:camera1 :name "Camera 1";
         :cameraType :Camera .
:camera2 :name "Camera 2";
         :price 150;
         :cameraType :Camera .
```

Example 1-119   DELETE/INSERT Operation on Default Graph

This example performs a DELETE/INSERT operation. The `:cameraType` of `:camera1` is updated to `:digitalCamera`.

```
-- Dataset before operation:
@prefix : <http://www.example.org/electronics/>
#Default graph
:camera1 :name "Camera 1";
         :cameraType :Camera .
:camera2 :name "Camera 2";
         :price 150;
         :cameraType :Camera .

-- Update operation:
BEGIN
  sem_apis.update_rdf_graph('electronics',
   'PREFIX : <http://www.example.org/electronics/>
    DELETE { :camera1 :cameraType ?type . }
    INSERT { :camera1 :cameraType :digitalCamera . }
    WHERE  { :camera1 :cameraType ?type . }',
   network_owner=>'RDFUSER', network_name=>'NET1');
END;
/

-- Dataset after operation:
@prefix : <http://www.example.org/electronics/>
#Default graph
:camera1 :name "Camera 1";
         :cameraType :digitalCamera .
:camera2 :name "Camera 2";
```

```
         :price 150;
         :cameraType :Camera .
```

Example 1-120 DELETE/INSERT Operation Involving Default Graph and Named Graph

Graphs can also be specified inside the DELETE and INSERT templates, as well as inside the WHERE clause. This example moves all triples corresponding to digital cameras from the default graph to the graph `:digitalCameras`.

```
-- Dataset before operation:
@prefix : <http://www.example.org/electronics/>
#Default graph
:camera1 :name "Camera 1";
         :cameraType :digitalCamera .
:camera2 :name "Camera 2";
         :price 150;
         :cameraType :Camera .
#Empty graph :digitalCameras

-- Update operation:
BEGIN
  sem_apis.update_rdf_graph('electronics',
   'PREFIX : <http://www.example.org/electronics/>
    DELETE { ?s ?p ?o }
    INSERT { graph :digitalCameras { ?s ?p ?o } }
    WHERE  { ?s :cameraType :digitalCamera .
             ?s ?p ?o }',
   network_owner=>'RDFUSER', network_name=>'NET1');
END;
/

-- Dataset after operation:
@prefix : <http://www.example.org/electronics/>
#Default graph
:camera2 :name "Camera 2";
         :price 150;
         :cameraType :Camera .
#Graph :digitalCameras
GRAPH :digitalCameras {
  :camera1 :name "Camera 1";
           :cameraType :digitalCamera .
}
```

Example 1-121 INSERT WHERE and DELETE WHERE Operations

One of either the DELETE template or the INSERT template can be omitted from a DELETE/INSERT operation. In addition, the template following DELETE can be omitted as a shortcut for using the WHERE pattern as the DELETE template. This example uses an INSERT WHERE statement to insert the contents of the `:digitalCameras` graph to the `:cameras` graph, and it uses a DELETE WHERE statement (with syntactic shortcut) to delete all contents of the `:cameras` graph.

```
-- INSERT WHERE
-- Dataset before operation:
```

```
@prefix : <http://www.example.org/electronics/>
#Default graph
:camera2 :name "Camera 2";
         :price 150;
         :cameraType :Camera .
#Graph :digitalCameras
GRAPH :digitalCameras {
  :camera1 :name "Camera 1";
           :cameraType :digitalCamera .
}
#Empty graph :cameras

-- Update operation:
BEGIN
  sem_apis.update_rdf_graph('electronics',
   'PREFIX : <http://www.example.org/electronics/>
    INSERT { graph :cameras { ?s ?p ?o } }
    WHERE  { graph :digitalCameras { ?s ?p ?o } }',
   network_owner=>'RDFUSER', network_name=>'NET1');
END;
/

-- Dataset after operation:
@prefix : <http://www.example.org/electronics/>
#Default graph
:camera2 :name "Camera 2";
         :price 150;
         :cameraType :Camera .
#Graph :digitalCameras
GRAPH :digitalCameras {
  :camera1 :name "Camera 1";
           :cameraType :digitalCamera .
}
#Graph :cameras
GRAPH :cameras {
  :camera1 :name "Camera 1";
           :cameraType :digitalCamera .
}

-- DELETE WHERE
-- Dataset before operation:
@prefix : <http://www.example.org/electronics/>
#Default graph
:camera2 :name "Camera 2";
         :price 150;
         :cameraType :Camera .
#Graph :digitalCameras
GRAPH :digitalCameras {
  :camera1 :name "Camera 1";
           :cameraType :digitalCamera .
}
#Graph :cameras
GRAPH :cameras {
  :camera1 :name "Camera 1";
           :cameraType :digitalCamera .
}
```

```
-- Update operation:
BEGIN
  sem_apis.update_rdf_graph('electronics',
   'PREFIX : <http://www.example.org/electronics/>
    DELETE WHERE { graph :cameras { ?s ?p ?o } }',
   network_owner=>'RDFUSER', network_name=>'NET1');
END;
/

-- Dataset after operation:
@prefix : <http://www.example.org/electronics/>
#Default graph
:camera2 :name "Camera 2";
         :price 150;
         :cameraType :Camera .
#Graph :digitalCameras
GRAPH :digitalCameras {
  :camera1 :name "Camera 1";
           :cameraType :digitalCamera .
}
#Empty graph :cameras
```

Example 1-122 COPY Operation

This example performs a COPY operation. All data from the default graph is inserted into the graph `:cameras`. Existing data from `:cameras`, if any, is removed before the insertion.

```
-- Dataset before operation:
@prefix : <http://www.example.org/electronics/>
#Default graph
:camera2 :name "Camera 2";
         :price 150;
         :cameraType :Camera .
#Graph :digitalCameras
GRAPH :digitalCameras {
  :camera1 :name "Camera 1";
           :cameraType :digitalCamera .
}
#Graph :cameras
GRAPH :cameras {
  :camera3 :name "Camera 3" .
}

-- Update operation:
BEGIN
  sem_apis.update_rdf_graph('electronics',
   'PREFIX : <http://www.example.org/electronics/>
    COPY DEFAULT TO GRAPH :cameras',
   network_owner=>'RDFUSER', network_name=>'NET1');
END;
/

-- Dataset after operation:
@prefix : <http://www.example.org/electronics/>
#Default graph
```

```
:camera2 :name "Camera 2";
         :price 150;
         :cameraType :Camera .
#Graph :digitalCameras
GRAPH :digitalCameras {
  :camera1 :name "Camera 1";
           :cameraType :digitalCamera .
}
#Graph :cameras
GRAPH :cameras {
  :camera2 :name "Camera 2";
           :price 150;
           :cameraType :Camera .
}
```

Example 1-123 ADD Operation

This example adds all the triples in the graph `:digitalCameras` to the graph `:cameras`.

```
-- Dataset before operation:
@prefix : <http://www.example.org/electronics/>
#Default graph
:camera2 :name "Camera 2";
         :price 150;
         :cameraType :Camera .
#Graph :digitalCameras
GRAPH :digitalCameras {
  :camera1 :name "Camera 1";
           :cameraType :digitalCamera .
}
#Graph :cameras
GRAPH :cameras {
  :camera2 :name "Camera 2";
           :price 150;
           :cameraType :Camera .
}

-- Update operation:
BEGIN
  sem_apis.update_rdf_graph('electronics',
   'PREFIX : <http://www.example.org/electronics/>
    ADD GRAPH :digitalCameras TO GRAPH :cameras',
   network_owner=>'RDFUSER', network_name=>'NET1');
END;
/

-- Dataset after operation:
@prefix : <http://www.example.org/electronics/>
#Default graph
:camera2 :name "Camera 2";
         :price 150;
         :cameraType :Camera .
#Graph :digitalCameras
GRAPH :digitalCameras {
  :camera1 :name "Camera 1";
           :cameraType :digitalCamera .
```

```
}
#Graph :cameras
GRAPH :cameras {
  :camera1 :name "Camera 1";
           :cameraType :digitalCamera .
  :camera2 :name "Camera 2";
           :price 150;
           :cameraType :Camera .
}
```

Example 1-124 MOVE Operation

This example moves all the triples in the graph `:digitalCameras` to the graph `:digCam`.

```
-- Dataset before operation:
@prefix : <http://www.example.org/electronics/>
#Default graph
:camera2 :name "Camera 2";
         :price 150;
         :cameraType :Camera .
#Graph :digitalCameras
GRAPH :digitalCameras {
  :camera1 :name "Camera 1";
           :cameraType :digitalCamera .
}
#Graph :cameras
GRAPH :cameras {
  :camera1 :name "Camera 1";
           :cameraType :digitalCamera .
  :camera2 :name "Camera 2";
           :price 150;
           :cameraType :Camera .
}
#Graph :digCam
GRAPH :digCam {
  :camera4 :cameraType :digCamera .
}

-- Update operation:
BEGIN
  sem_apis.update_rdf_graph('electronics',
   'PREFIX : <http://www.example.org/electronics/>
    MOVE GRAPH :digitalCameras TO GRAPH :digCam',
   network_owner=>'RDFUSER', network_name=>'NET1');
END;
/

-- Dataset after operation:
@prefix : <http://www.example.org/electronics/>
#Default graph
:camera2 :name "Camera 2" .
         :camera2 :price 150 .
         :camera2 :cameraType :Camera .
#Empty graph :digitalCameras
#Graph :cameras
GRAPH :cameras {
```

```
  :camera1 :name "Camera 1";
           :cameraType :digitalCamera .
  :camera2 :name "Camera 2";
           :price 150;
           :cameraType :Camera .
}
#Graph :digCam
GRAPH :digCam {
  :camera1 :name "Camera 1";
           :cameraType :digitalCamera .
}
```

Example 1-125 CLEAR Operation

This example performs a CLEAR operation, deleting all the triples in the default graph. Because empty graphs are not stored in the RDF graph, the CLEAR operation always succeeds and is equivalent to a DROP operation. (For the same reason, the CREATE operation has no effect on the RDF graph.)

```
-- Dataset before operation:
@prefix : <http://www.example.org/electronics/>
#Default graph
:camera2 :name "Camera 2";
         :price 150;
         :cameraType :Camera .
#Empty graph :digitalCameras
#Graph :cameras
GRAPH :cameras {
  :camera1 :name "Camera 1";
           :cameraType :digitalCamera
  :camera2 :name "Camera 2";
           :price 150;
           :cameraType :Camera .
}
#Graph :digCam
GRAPH :digCam {
  :camera1 :name "Camera 1";
           :cameraType :digitalCamera .
}

-- Update operation:
BEGIN
  sem_apis.update_rdf_graph('electronics',
   'CLEAR DEFAULT ',
   network_owner=>'RDFUSER', network_name=>'NET1');
END;
/

-- Dataset after operation:
@prefix : <http://www.example.org/electronics/>
#Empty Default graph
#Empty graph :digitalCameras
#Graph :cameras
GRAPH :cameras {
  :camera1 :name "Camera 1";
           :cameraType :digitalCamera .
```

```
  :camera2 :name "Camera 2";
           :price 150;
           :cameraType :Camera .
}
#Graph :digCam
GRAPH :digCam {
  :camera1 :name "Camera 1";
           :cameraType :digitalCamera .
}
```

Example 1-126   LOAD Operation

N-Triple, N-Quad, Turtle, and Trig files can be loaded from the local file system using the LOAD operation. Note that the simpler N-Triple, and N-Quad formats can be loaded faster than Turtle and Trig. An optional INTO clause can be used to load the file into a specific named graph. To perform a LOAD operation, the user must either (1) have CREATE ANY DIRECTORY and DROP ANY DIRECTORY privileges or (2) supply the name of an existing directory object in the `options` parameter of UPDATE_RDF_GRAPH. This example loads the /home/oracle/example.nq N-Quad file into an RDF graph..

Note that the use of an INTO clause with an N-Quad or Trig file will override any named graph information in the file. In this example, `INTO GRAPH :cameras` overrides `:myGraph` for the first quad, so the subject, property, object triple component of this quad is inserted into the `:cameras` graph instead.

```
-- Datafile: /home/oracle/example.nq
<http://www.example.org/electronics/camera3> <http://www.example.org/
electronics/name> "Camera 3" <http://www.example.org/electronics/myGraph> .
<http://www.example.org/electronics/camera3> <http://www.example.org/
electronics/price> "125"^^<http://www.w3.org/2001/XMLSchema#decimal> .

-- Dataset before operation:
#Graph :cameras
GRAPH :cameras {
  :camera1 :name "Camera 1";
           :cameraType :digitalCamera .
  :camera2 :name "Camera 2";
           :price 150;
           :cameraType :Camera .
}
#Graph :digCam
GRAPH :digCam {
  :camera1 :name "Camera 1";
           :cameraType :digitalCamera .
}

-- Update operation:
CREATE OR REPLACE DIRECTORY MY_DIR AS '/home/oracle';

BEGIN
  sem_apis.update_rdf_graph('electronics',
   'PREFIX : <http://www.example.org/electronics/>
    LOAD <file:///example.nq> INTO GRAPH :cameras',
   options=>'LOAD_DIR={MY_DIR}',
   network_owner=>'RDFUSER', network_name=>'NET1');
END;
```

```
END;
/

-- Dataset after operation:
@prefix : <http://www.example.org/electronics/>
#Graph :cameras
GRAPH :cameras {
  :camera1 :name "Camera 1";
           :cameraType :digitalCamera .
  :camera2 :name "Camera 2";
           :price 150;
           :cameraType :Camera .
  :camera3 :name "Camera 3";
           :price 125.
}
#Graph :digCam
GRAPH :digCam {
  :camera1 :name "Camera 1";
           :cameraType :digitalCamera .
}
```

Several files under the same directory can be loaded in parallel with a single LOAD operation. To specify extra N-Triple or N-Quad files to be loaded, you can use the LOAD_OPTIONS hint. The degree of parallelism for the load can be specified with `PARALLEL(n)` in the `options` string.. The following example shows how to load the files `/home/oracle/example1.nq`, `/home/oracle/example2.nq`, and `/home/oracle/example3.nq` into an RDF graph. A degree of parallelism of 3 is used for this example.

```
BEGIN
  sem_apis.update_rdf_graph('electronics',
   'PREFIX : <http://www.example.org/electronics/>
    LOAD <file:///example1.nq>',
   options=> ' PARALLEL(3) LOAD_OPTIONS={ example2.nq example3.nq }
LOAD_DIR={MY_DIR} ',
   network_owner=>'RDFUSER', network_name=>'NET1' );
END;
/
```

Related subtopics:

- Tuning the Performance of SPARQL Update Operations
- Transaction Management with SPARQL Update Operations
- Support for Bulk Operations
- Setting UPDATE_RDF_GRAPH Options at the Session Level
- Load Operations: Special Considerations for SPARQL Update
- Long Literals: Special Considerations for SPARQL Update
- Blank Nodes: Special Considerations for SPARQL Update

1.15.1 Tuning the Performance of SPARQL Update Operations

In some cases it may be necessary to tune the performance of SPARQL Update operations. Because SPARQL Update operations involve executing one or more SPARQL queries based on the WHERE clause in the UPDATE statement, the Best Practices for Query Performance also apply to SPARQL Update operations. The following considerations also apply:

- Delete operations require an appropriate index on the application table (associated with the `apply_model` parameter in SEM_APIS.UPDATE_RDF_GRAPH) for good performance. Assuming an application table named APP_TAB with the SDO_RDF_TRIPLE_S column named TRIPLE, an index similar to the following is recommended (this is the same index used by RDF Graph Support for Apache Jena):

```
-- Application table index for
--  (graph_id, subject_id, predicate_id, canonical_object_id)
CREATE INDEX app_tab_idx ON app_tab app (
  BITAND(app.triple.rdf_m_id,79228162514264337589248983040)/4294967296,
  app.triple.rdf_s_id,
  app.triple.rdf_p_id,
  app.triple.rdf_c_id)
COMPRESS;
```

- Performance-related SEM_MATCH options can be passed to the `match_options` parameter of SEM_APIS.UPDATE_RDF_GRAPH, and performance-related options such as PARALLEL and DYNAMIC_SAMPLING can be specified in the `options` parameter of that procedure. The following example uses the options parameter to specify a degree of parallelism of 4 and an optimizer dynamic sampling level of 6 for the update. In addition, the example uses ALLOW_DUP=T as a match option when matching against the RDF graph collection VM1.

```
BEGIN
  sem_apis.update_rdf_graph(
   'electronics',
   'PREFIX : <http://www.example.org/electronics/>
    INSERT { graph :digitalCameras { ?s ?p ?o } }
    WHERE  { ?s :cameraType :digitalCamera .
             ?s ?p ?o }',
   match_models=>sem_models('VM1'),
   match_models=>sem_models('VM1'),
   match_options=>' ALLOW_DUP=T ',
   options=>' PARALLEL(4) DYNAMIC_SAMPLING(6) ',
   network_owner=>'RDFUSER', network_name=>'NET1');
END;
/
```

- Inline Query Optimizer Hints can be specified in the WHERE clause. The following example extends the preceding example by using the HINT0 hint in the WHERE clause and the FINAL_VALUE_NL hint in the `match_options` parameter.

```
BEGIN
  sem_apis.update_rdf_graph(
   'electronics',
   'PREFIX : <http://www.example.org/electronics/>
    INSERT { graph :digitalCameras { ?s ?p ?o } }
    WHERE  { # HINT0={ LEADING(t0 t1) USE_NL(t0 t1)
             ?s :cameraType :digitalCamera .
             ?s ?p ?o }',
   match_models=>sem_models('VM1'),
   match_options=>' ALLOW_DUP=T FINAL_VALUE_NL ',
   options=>' PARALLEL(4) DYNAMIC_SAMPLING(6) ',
```

```
      network_owner=>'RDFUSER', network_name=>'NET1');
   END;
   /
```

1.15.2 Transaction Management with SPARQL Update Operations

You can exercise some control over the number of transactions used and whether they are automatically committed by a SEM_APIS.UPDATE_RDF_GRAPH operation.

By default, the SEM_APIS.UPDATE_RDF_GRAPH procedure executes in a single transaction that is either committed upon successful completion or rolled back if an error occurs. For example, the following call executes three update operations (separated by semicolons) in a single transaction:

```
BEGIN
  sem_apis.update_rdf_graph('electronics',
   'PREFIX elec: <http://www.example.org/electronics/>
    PREFIX ecom: <http://www.example.org/ecommerce/>
    # insert camera data
    INSERT DATA {
      elec:camera1 elec:name "Camera 1" .
      elec:camera1 elec:price 120 .
      elec:camera1 elec:cameraType elec:DigitalCamera .
      elec:camera2 elec:name "Camera 2" .
      elec:camera2 elec:price 150 .
      elec:camera2 elec:cameraType elec:DigitalCamera . };
    # insert ecom:price triples
    INSERT { ?c  ecom:price ?p }
    WHERE  { ?c  elec:price ?p };
    # delete elec:price triples
    DELETE WHERE { ?c elec:price ?p }',
   network_owner=>'RDFUSER', network_name=>'NET1');
END;
/

PL/SQL procedure successfully completed.
```

By contrast, the following example uses three separate SEM_APIS.UPDATE_RDF_GRAPH calls to execute the same three update operations in three separate transactions:

```
BEGIN
  sem_apis.update_rdf_graph('electronics',
   'PREFIX elec: <http://www.example.org/electronics/>
    PREFIX ecom: <http://www.example.org/ecommerce/>
    # insert camera data
    INSERT DATA {
      elec:camera1 elec:name "Camera 1" .
      elec:camera1 elec:price 120 .
      elec:camera1 elec:cameraType elec:DigitalCamera .
      elec:camera2 elec:name "Camera 2" .
      elec:camera2 elec:price 150 .
      elec:camera2 elec:cameraType elec:DigitalCamera . }',
   network_owner=>'RDFUSER', network_name=>'NET1');
END;
```

```
PL/SQL procedure successfully completed.

BEGIN
  sem_apis.update_rdf_graph('electronics',
   'PREFIX elec: <http://www.example.org/electronics/>
    PREFIX ecom: <http://www.example.org/ecommerce/>
    # insert ecom:price triples
    INSERT { ?c  ecom:price ?p }
    WHERE  { ?c  elec:price ?p }',
   network_owner=>'RDFUSER', network_name=>'NET1');
END;
/

PL/SQL procedure successfully completed.

BEGIN
  sem_apis.update_rdf_graph('electronics',
   'PREFIX elec: <http://www.example.org/electronics/>
    PREFIX ecom: <http://www.example.org/ecommerce/>
    # insert elec:price triples
    DELETE WHERE { ?c elec:price ?p }',
   network_owner=>'RDFUSER', network_name=>'NET1');
END;
/

PL/SQL procedure successfully completed.
```

The `AUTOCOMMIT=F` option can be used to prevent separate transactions for each SEM_APIS.UPDATE_RDF_GRAPH call. With this option, transaction management is the responsibility of the caller. The following example shows how to execute the update operations in the preceding example as a single transaction instead of three separate ones.

```
BEGIN
  sem_apis.update_rdf_graph('electronics',
   'PREFIX elec: <http://www.example.org/electronics/>
    PREFIX ecom: <http://www.example.org/ecommerce/>
    # insert camera data
    INSERT DATA {
      elec:camera1 elec:name "Camera 1" .
      elec:camera1 elec:price 120 .
      elec:camera1 elec:cameraType elec:DigitalCamera .
      elec:camera2 elec:name "Camera 2" .
      elec:camera2 elec:price 150 .
      elec:camera2 elec:cameraType elec:DigitalCamera . }',
   options=>' AUTOCOMMIT=F ',
   network_owner=>'RDFUSER', network_name=>'NET1');
END;
/

PL/SQL procedure successfully completed.

BEGIN
  sem_apis.update_rdf_graph('electronics',
   'PREFIX elec: <http://www.example.org/electronics/>
    PREFIX ecom: <http://www.example.org/ecommerce/>
```

```
     # insert ecom:price triples
     INSERT { ?c  ecom:price ?p }
     WHERE  { ?c  elec:price ?p }',
    options=>' AUTOCOMMIT=F ',
    network_owner=>'RDFUSER', network_name=>'NET1');
END;
/

PL/SQL procedure successfully completed.

BEGIN
  sem_apis.update_rdf_graph('electronics',
    'PREFIX elec: <http://www.example.org/electronics/>
     PREFIX ecom: <http://www.example.org/ecommerce/>
     # insert elec:price triples
     DELETE WHERE { ?c elec:price ?p }',
    options=>' AUTOCOMMIT=F ',
    network_owner=>'RDFUSER', network_name=>'NET1');
END;
/

PL/SQL procedure successfully completed.

COMMIT;

Commit complete.
```

However, the following cannot be used with the `AUTOCOMMIT=F` option:

- Bulk operations (`FORCE_BULK=T`, `DEL_AS_INS=T`)
- `LOAD` operations
- Materialization of intermediate data (`STREAMING=F`)
- Transaction Isolation Levels

1.15.2.1 Transaction Isolation Levels

Oracle Database supports three different transaction isolation levels: read committed, serializable, and read-only.

Read committed isolation level is the default. Queries in a transaction using this isolation level see only data that was committed before the query – not the transaction – began and any changes made by the transaction itself. This isolation level allows the highest degree of concurrency.

Serializable isolation level queries see only data that was committed before the transaction began and any changes made by the transaction itself.

Read-only isolation level behaves like serializable isolation level but data cannot be modified by the transaction.

SEM_APIS.UPDATE_RDF_GRAPH supports read committed and serializable transaction isolation levels, and read committed is the default. SPARQL UPDATE operations are processed in the following basic steps.

1. A query is executed to obtain a set of triples to be deleted.

2. A query is executed to obtain a set of triples to be inserted.
3. Triples obtained in Step 1 are deleted.
4. Triples obtained in Step 2 are inserted.

With the default read committed isolation level, the underlying triple data may be modified by concurrent transactions, so each step may see different data. In addition, changes made by concurrent transactions will be visible to subsequent update operations within the same SEM_APIS.UPDATE_RDF_GRAPH call. Note that steps 1 and 2 happen as a single step when using materialization of intermediate data (`STREAMING=F`), so underlying triple data cannot be modified between steps 1 and 2 with this option. See Support for Bulk Operations for more information about materialization of intermediate data.

Serializable isolation level can be used by specifying the `SERIALIZABLE=T` option. In this case, each step will only see data that was committed before the update RDF graph operation began, and multiple update operations executed in a single SEM_APIS.UPDATE_RDF_GRAPH call will not see modifications made by concurrent update operations in other transactions. However, ORA-08177 errors will be raised if a SEM_APIS.UPDATE_RDF_GRAPH execution tries to update triples that were modified by a concurrent transaction. When using `SERIALIZABLE=T`, the application should detect and handle ORA-08177 errors (for example, retry the update command if it could not be serialized on the first attempt).

The following cannot be used with the `SERIALIZABLE=T` option:

- Bulk operations (`FORCE_BULK=T`, `DEL_AS_INS=T`)
- `LOAD` operations
- Materialization of intermediate data (`STREAMING=F`)

1.15.3 Support for Bulk Operations

SEM_APIS.UPDATE_RDF_GRAPH supports bulk operations for efficient execution of large updates. The following options are provided; however, when using any of these bulk operations, serializable isolation (`SERIALIZABLE=T`) and autocommit false (`AUTOCOMMMIT=F`) cannot be used.

- Materialization of Intermediate Data (STREAMING=F)
- Using SEM_APIS.BULK_LOAD_RDF_GRAPH
- Using Delete as Insert (DEL_AS_INS=T)

1.15.3.1 Materialization of Intermediate Data (STREAMING=F)

By default, SEM_APIS.UPDATE_RDF_GRAPH executes two queries for a basic DELETE INSERT SPARQL Update operation: one query to find triples to delete and one query to find triples to insert. For some update operations with WHERE clauses that are expensive to evaluate, executing two queries may not give the best performance. In these cases, executing a single query for the WHERE clause, materializing the results, and then using the materialized results to construct triples to delete and triples to insert may give better performance. This approach incurs overhead from a DDL operation, but overall performance is likely to be better for complex update statements.

The following example update using this option (`STREAMING=F`). Note that `STREAMING=F` is not allowed with serializable isolation (`SERIALIZABLE=T`) or autocommit false (`AUTOCOMMIT=F`).

```
BEGIN
  sem_apis.update_rdf_graph('electronics',
   'PREFIX : <http://www.example.org/electronics/>
    DELETE { ?s ?p ?o }
    INSERT { graph :digitalCameras { ?s ?p ?o } }
    WHERE  { ?s :cameraType :digitalCamera .
             ?s ?p ?o }',
  options=>' STREAMING=F ',
  network_owner=>'RDFUSER', network_name=>'NET1');
END;
/
```

1.15.3.2 Using SEM_APIS.BULK_LOAD_RDF_GRAPH

For updates that insert a large number of triples (such as tens of thousands), the default approach of incremental DML on the application table may not give acceptable performance. In such cases, the `FORCE_BULK=T` option can be specified so that SEM_APIS.BULK_LOAD_RDF_GRAPH is used instead of incremental DML.

However, not all update operations can use this optimization. The `FORCE_BULK=T` option is only allowed for a SEM_APIS.UPDATE_RDF_GRAPH call with either a single ADD operation or a single INSERT WHERE operation. The use of SEM_APIS.BULK_LOAD_RDF_GRAPH forces a series of commits and autonomous transactions, so the `AUTOCOMMIT=F` and `SERIALIZABLE=T` options are not allowed with `FORCE_BULK=T`. In addition, bulk load cannot be used with `CLOB_UPDATE_SUPPORT=T`.

SEM_APIS.BULK_LOAD_RDF_GRAPH allows various customizations through its `flags` parameter. SEM_APIS.UPDATE_RDF_GRAPH supports the `BULK_OPTIONS={ OPTIONS_STRING }` flag so that `OPTIONS_STRING` can be passed into the `flags` input of SEM_APIS.BULK_LOAD_RDF_GRAPH to customize bulk load options. The following example shows a SEM_APIS.UPDATE_RDF_GRAPH invocation using the `FORCE_BULK=T` option and `BULK_OPTIONS` flag.

```
BEGIN
  sem_apis.update_rdf_graph('electronics',
   'PREFIX elec: <http://www.example.org/electronics/>
    PREFIX ecom: <http://www.example.org/ecommerce/>
    INSERT { ?c  ecom:price ?p }
    WHERE  { ?c  elec:price ?p }',
   options=>' FORCE_BULK=T BULK_OPTIONS={  parallel=4
parallel_create_index }',
   network_owner=>'RDFUSER', network_name=>'NET1');
END;
/
```

1.15.3.3 Using Delete as Insert (DEL_AS_INS=T)

For updates that delete a large number of triples (such as tens of thousands), the default approach of incremental DML on the application table may not give acceptable performance. For such cases, the `DEL_AS_INS=T` option can be specified. With this option, a large delete operation is implemented as INSERT, TRUNCATE, and EXCHANGE PARTITION operations.

The use of DEL_AS_INS=T causes a series of commits and autonomous transactions, so this option cannot be used with SERIALIZABLE=T or AUTOCOMMIT=F. In addition, this option can only be used with SEM_APIS.UPDATE_RDF_GRAPH calls that involve a single DELETE WHERE operation, a single DROP operation, or a single CLEAR operation.

Delete as insert internally uses SEM_APIS.MERGE_RDF_GRAPHS during intermediate operations. The string OPTIONS_STRING from the MM_OPTIONS={ OPTIONS_STRING } flag can be specified to customize options for merging. The following example shows a SEM_APIS.UPDATE_RDF_GRAPH invocation using the DEL_AS_INS=T option and MM_OPTIONS flag.

```
BEGIN
  sem_apis.update_rdf_graph('electronics',
   'CLEAR NAMED',
   options=>' DEL_AS_INS=T MM_OPTIONS={  dop=4 } ',
   network_owner=>'RDFUSER', network_name=>'NET1');
END;
/
```

1.15.4 Setting UPDATE_RDF_GRAPH Options at the Session Level

Some settings that affect the SEM_APIS.UPDATE_RDF_GRAPH procedure's behavior can be modified at the session level through the use of the special MDSYS.SDO_SEM_UPDATE_CTX.SET_PARAM procedure. The following options can be set to true or false at the session level: autocommit, streaming, strict_bnode, and clob_support.

The MDSYS.SDO_SEM_UPDATE_CTX contains the following subprograms to get and set SEM_APIS.UPDATE_RDF_GRAPH parameters at the session level:

```
SQL> describe mdsys.sdo_sem_update_ctx
FUNCTION GET_PARAM RETURNS VARCHAR2
 Argument Name                  Type                    In/Out Default?
 ------------------------------ ----------------------- ------ --------
 NAME                           VARCHAR2                IN
PROCEDURE SET_PARAM
 Argument Name                  Type                    In/Out Default?
 ------------------------------ ----------------------- ------ --------
 NAME                           VARCHAR2                IN
 VALUE                          VARCHAR2                IN
```

The following example causes all subsequent calls to the SEM_APIS.UPDATE_RDF_GRAPH procedure to use the AUTOCOMMIT=F setting, until the end of the session or the next call to SEM_APIS.UPDATE_RDF_GRAPH that specifies a different autocommit value.

```
begin
  mdsys.sdo_sem_update_ctx.set_param('autocommit','false');
end;
/
```

1.15.5 Load Operations: Special Considerations for SPARQL Update

The format of the file to load affects the amount of parallelism that can be used during the load process. Load operations have two phases:

1. Loading from the file system to a staging table
2. Calling SEM_APIS.BULK_LOAD_RDF_GRAPH to load from a staging table into an RDF graph

All supported data formats can use parallel execution in phase 2, but only N-Triple and N-Quad formats can use parallel execution in phase 1. In addition, if a load operation is interrupted during phase 2 after the staging table has been fully populated, loading can be resumed with the `RESUME_LOAD=T` keyword in the `options` parameter.

Load operations for RDF documents that contain object values longer than `NETWORK_MAX_STRING_SIZE` bytes may require additional operations. Load operations on Turtle and Trig documents will automatically load all triples/quads regardless of object value size. However, load operations on N-Triple and N-Quad documents will only load triples/quads with object values that are less than `NETWORK_MAX_STRING_SIZE` bytes in length. For N-Triple and N-Quad data, a second load operation should be issued with the `LOAD_CLOB_ONLY=T` option to also load triples/quads with object values larger than `NETWORK_MAX_STRING_SIZE` bytes.

Loads from Unix named pipes are only supported for N-Triple and N-Quad formats. Turtle and Trig files should be uncompressed, physical files.

Unicode characters are handled differently depending on the format of the RDF file to load. Unicode characters in N-Triple and N-Quad files should be escaped as `\u<HEX><HEX><HEX><HEX>` or `\U<HEX><HEX><HEX><HEX><HEX><HEX><HEX><HEX>` using the hex value of the Unicode codepoint value. Turtle and Trig files do not require Unicode escaping and can be directly loaded with unescaped Unicode values.

Example 1-127 Short and Long Literal Load for N-Quad Data

```
BEGIN
  -- short literal load
  sem_apis.update_rdf_graph('electronics',
   'PREFIX : <http://www.example.org/electronics/>
    LOAD <file:///example1.nq>',
   options=> ' LOAD_DIR={MY_DIR} ',
   network_owner=>'RDFUSER', network_name=>'NET1');

  -- long literal load
  sem_apis.update_rdf_graph('electronics',
   'PREFIX : <http://www.example.org/electronics/>
    LOAD <file:///example1.nq>',
   options=> ' LOAD_DIR={MY_DIR} LOAD_CLOB_ONLY=T ',
   network_owner=>'RDFUSER', network_name=>'NET1');
END;
/
```

1.15.6 Long Literals: Special Considerations for SPARQL Update

By default, SPARQL Update operations do not manipulate values longer than `NETWORK_MAX_STRING_SIZE` bytes. To enable long literals support, specify `CLOB_UPDATE_SUPPORT=T` in the options parameter with the SEM_APIS.UPDATE_RDF_GRAPH procedure.

Bulk load does not work for long literals; the `FORCE_BULK=T` option is ignored when used with the `CLOB_UPDATE_SUPPORT=T` option.

1.15.7 Blank Nodes: Special Considerations for SPARQL Update

Some update operations only affect the graph of a set of RDF triples. Specifically, these operations are ADD, COPY and MOVE. For example, the MOVE operation example in Support for SPARQL Update Operations on an RDF Graph can be performed only updating triples having `:digitalCameras` as the graph. However, the performance of such operations can be improved by using ID-only operations over the RDF graph. To run a large ADD, COPY, or MOVE operation as an ID-only operation, you can specify the `STRICT_BNODE=F` hint in the `options` parameter for the SEM_APIS.UPDATE_RDF_GRAPH procedure.

ID-only operations may lead to incorrect blank nodes, however, because no two graphs should share the same blank node. RDF graph uses a blank node prefixing scheme based on the model (RDF graph) and named graph combination that contains a blank node. These prefixes ensure that blank node identifiers are unique across models (RDF graphs) and named graphs. An ID-only approach for ADD, COPY, and UPDATE operations does not update blank node prefixes.

Example 1-128   ID-Only Update Causing Incorrect Blank Node Values

The update in the following example leads to the same blank node subject for both triples in graphs `:cameras` and `:cameras2`. This can be verified running the provided SEM_MATCH query.

```
BEGIN
  sem_apis.update_rdf_graph('electronics',
   'PREFIX : <http://www.example.org/electronics/>
    INSERT DATA {
       GRAPH :cameras { :camera2 :owner _:bn1 .
                        _:bn1 :name "Axel" }
    };
    COPY :cameras TO :cameras2',
   options=>' STRICT_BNODE=F ',
   network_owner=>'RDFUSER', network_name=>'NET1');
END;
/

SELECT count(s)
FROM TABLE( SEM_MATCH('
  PREFIX : <http://www.example.org/electronics/>
  SELECT *
  WHERE { { graph :cameras  {?s :name "Axel"  } }
          { graph :cameras2 {?s :name "Axel"  } } }
', sem_models('electronics'),null,null,null,null,' STRICT_DEFAULT=T ',
null, null, 'RDFUSER', 'NET1'));
```

To avoid such errors, you should specify the `STRICT_BNODE=F` hint in the `options` parameter for the SEM_APIS.UPDATE_RDF_GRAPH procedure only when you are sure that blank nodes are not involved in the ADD, COPY, or MOVE update operation.

However, ADD, COPY, and MOVE operations on large graphs with the `STRICT_BNODE=F` option may run significantly faster than they would run using the default method. If you need to run a series of ID-only updates, another option is to use the `STRICT_BNODE=F` option, and then execute the SEM_APIS.CLEANUP_BNODES procedure at the end. This approach resets the prefix of all blank nodes in a given RDF graph, which effectively corrects ("cleans up") all erroneous blank node labels.

Note that this two-step strategy should not be used with a small number of ADD, COPY, or MOVE operations. Performing a few operations using the default approach will execute faster than running a few ID-only operations and then executing the SEM_APIS.CLEANUP_BNODES procedure.

The following example corrects blank nodes in the RDF graph named `electronics`.

```
EXECUTE sem_apis.cleanup_bnodes('electronics');
```

1.16 RDF Support for Oracle Database In-Memory

RDF can use the in-memory Oracle Database In-Memory suite of features, including in-memory column store, to improve performance for real-time analytics and mixed workloads.

After Database In-Memory setup, the RDF in-memory loading can be performed using the SEM_APIS.ENABLE_INMEMORY procedure. This requires an administrative privilege and affects the entire RDF network. It loads frequently used columns from the RDF_LINK$ and RDF_VALUE$ tables into memory.

After this procedure is executed, RDF in-memory virtual columns can be loaded into memory. This is done at the RDF graph collection level: when an RDF graph collection is created, the in-memory option can be specified in the call to SEM_APIS.CREATE_RDF_GRAPH_COLLECTION.

You can also enable and disable in-memory population of RDF data for specified RDF graphs and inferred graphs by using the SEM_APIS.ENABLE_INMEMORY_FOR_RDF_GRAPH, SEM_APIS.ENABLE_INMEMORY_FOR_INF_GRAPH, SEM_APIS.DISABLE_INMEMORY_FOR_RDF_GRAPH, and SEM_APIS.DISABLE_INMEMORY_FOR_INF_GRAPH procedures.

Note:

To use RDF with Oracle Database In-Memory, you must understand how to enable and configure Oracle Database In-Memory, as explained in *Oracle Database In-Memory Guide*.

- Enabling Oracle Database In-Memory for RDF
- Using In-Memory Virtual Columns with RDF
- Using Invisible Indexes with Oracle Database In-Memory

1.16.1 Enabling Oracle Database In-Memory for RDF

To load RDF data into memory, the `compatibility` must be set to 12.2 or later, and the `inmemory_size` value must be at least 100MB. The RDF network can then be loaded into memory using the SEM_APIS.ENABLE_INMEMORY procedure.

Before you use RDF data in memory, you should verify that the data is loaded into memory:

```
SQL> select pool, alloc_bytes, used_bytes, populate_status from V$INMEMORY_AREA;
POOL                  ALLOC_BYTES USED_BYTES POPULATE_STATUS
-------------------------- ----------- ---------- --------------------------
1MB POOL                5.0418E+10 4.4603E+10 DONE
64KB POOL                3202088960      9568256   DONE
```

If the POPULATE_STATUS value is DONE, the RDF data has been fully loaded into memory.

To check if RDF data in memory is used, search for ‘TABLE ACCESS INMEMORY FULL’ in the execution plan:

```
-----------------------------------------------------------------------------------------
---------------------------------------------------
| Id  | Operation              | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
Pstart| Pstop |    TQ  |IN-OUT| PQ Distrib |
-----------------------------------------------------------------------------------------
---------------------------------------------------
|   0 | SELECT STATEMENT          |          |     1 |    13 |   580  (60)| 00:00:01
|       |       |        |      |         |
|   1 |  VIEW                  |          |     1 |    13 |   580  (60)| 00:00:01
|       |       |        |      |         |
|   2 |   VIEW                 |          |     1 |    13 |   580  (60)| 00:00:01
|       |       |        |      |         |
|   3 |    SORT AGGREGATE          |          |     1 |    16 |         |
|       |       |        |      |         |
|   4 |     PX COORDINATOR          |          |       |       |         |
|       |       |        |      |         |
|   5 |      PX SEND QC (RANDOM)      | :TQ10000  |     1 |    16 |         |
|       |       |  Q1,00 | P->S | QC (RAND)  |
|   6 |       SORT AGGREGATE          |          |     1 |    16 |         |
|       |       |  Q1,00 | PCWP |         |
|   7 |        PX BLOCK ITERATOR      |          |   242M|  3697M|   580  (60)| 00:00:01
|KEY(I) |KEY(I) |  Q1,00 | PCWC |         |
|   8 |      TABLE ACCESS INMEMORY FULL| RDF_LINK$ |   242M|  3697M|   580  (60)|
00:00:01 |KEY(I) |KEY(I) |  Q1,00 | PCWP |         |
-----------------------------------------------------------------------------------------
---------------------------------------------------
```

To disable in-memory population of RDF data, use the SEM_APIS.DISABLE_INMEMORY procedure.

1.16.2 Using In-Memory Virtual Columns with RDF

In addition to RDF data in memory, RDF in-memory virtual columns can be used to load lexical values for RDF terms in the RDF_LINK$ table into memory. To load the RDF in-memory virtual columns, you must first execute SEM_APIS.ENABLE_INMEMORY with administrative privileges, setting the inmemory_virtual_columns parameter to ENABLE. The in-memory virtual columns are created in the RDF_LINK$ table and loaded into memory at the RDF graph collection level.

To load the virtual columns into memory, use the option ‘PXN=F INMEMORY=T’ in the call to SEM_APIS.CREATE_RDF_GRAPH_COLLECTION. For example (assuming a schema-private network named NET1 owned by a database user named RDFUSER):

```
EXECUTE  SEM_APIS.CREATE_RDF_GRAPH_COLLECTION
('vm2',SEM_MODELS('lubm1k','univbench'),SEM_RULEBASES
('owl2rl'),options=>'PXN=F INMEMORY=T', network_owner=>'RDFUSER',
network_name=>'NET1');
```

You can check for in-memory RDF graph collections by examining the SEM_MODEL$ view, where the INMEMORY column is set to T for an in-memory graph collection.

The in-memory RDF graph collection removes the need for joins with the RDF_VALUE$ table. To check the usage of in-memory RDF graph collections, use the same commands in Enabling Oracle Database In-Memory for RDF.

For best performance, fully populate the in-memory virtual columns before any query is processed, because unpopulated virtual columns are assembled at run time and this overhead may impair performance.

1.16.3 Using Invisible Indexes with Oracle Database In-Memory

Sometimes, inconsistent query performance may result due to the use of indexes. If you want consistent performance across different workloads, even though it may mean negating some performance gains that normally result from indexing, you can make the RDF network indexes **invisible** so that the query execution is done by pure memory scans. The following example makes the RDF network indexes invisible in a schema-private network named NET1 owned by a database user named RDFUSER:

```
EXECUTE SEM_APIS.ALTER_RDF_INDEXES('VISIBILITY','N',
network_owner=>'RDFUSER', network_name=>'NET1');
```

To make the RDF network indexes visible again, use the following

```
EXECUTE SEM_APIS.ALTER_RDF_INDEXES('VISIBILITY','Y',
network_owner=>'RDFUSER', network_name=>'NET1');
```

> **Note:**
>
> RDF_VALUE$ indexes must be visible so that Oracle Database can efficiently look up VALUE_IDs for query constants at compile time.

For an explanation of invisible and unusable indexes, see *Oracle Database Administrator's Guide*.

1.17 RDF Support for Materialized Join Views

The most frequently used joins in RDF queries are subject-subject and subject-object joins. To enhance the RDF query performance, you can create materialized join views on those two columns.

Materialized join views can be created on a single RDF graph, or on more than one graphs by creating an RDF graph collection with the `'ALLOW_DUP=T'` option, and then creating the materialized join view on that RDF graph collection. All materialized views are owned by the network owner. (To create a materialized join view, use the SEM_APIS.CREATE_MATERIALIZED_VIEW procedure.)

The materialized views are compressed by default, and in-memory can be enabled if the IMDB option is installed. Two materialized views are created on subject-subject join (SS-join) and subject-object join (SO-join) between two tables named, for example, T0 and T1, and all G,S,P,O values are fetched by a deterministic function using the IDs. The values can optionally be defined as a virtual column. In other words, only G,S,P,O IDs for both T0 and T1 are real columns, and the rest are virtual columns. It is recommended that the virtual columns be used with in-memory virtual column enabled, so that the values are materialized in memory if the IMDB option is installed.

A bitmap index can be created on a single column in the materialized view. The materialized view columns are named as follows in each table in the join:

- Graph ID: G
- Subject ID: S
- Predicate ID: P
- Object ID: O
- Graph name: GV
- Subject name: SV
- Predicate name: PV
- Object name: OV
- value type: $RDFVTYP
- literal type: $RDFLTYP
- language type: $RDFLANG
- order_type: $RDFORDT
- order_num: $RDFORDN
- order_date: $RDFORDD

For example, if a materialized view named MVX is created, the following join views are created:

```
SS-join (MVX$SS) and SO-join (MVX$SO)

MVX$SS(T0G, T0S, T0P, T0O, T1G, T1S, T1P, T1O,
  T0GV, T0G$RDFVTYP, T0G$RDFLTYP, T0G$RDFLANG, T0G$RDFORDT, T0G$RDFORDN, T0G$RDFORDD
  T0SV, T0S$RDFVTYP, T0S$RDFLTYP, T0S$RDFLANG, T0S$RDFORDT, T0S$RDFORDN, T0S$RDFORDD
  T0PV, T0P$RDFVTYP, T0P$RDFLTYP, T0P$RDFLANG, T0P$RDFORDT, T0P$RDFORDN, T0P$RDFORDD
  T0OV, T0O$RDFVTYP, T0O$RDFLTYP, T0O$RDFLANG, T0O$RDFORDT, T0O$RDFORDN, T0O$RDFORDD
  T1GV, T1G$RDFVTYP, T1G$RDFLTYP, T1G$RDFLANG, T1G$RDFORDT, T1G$RDFORDN, T1G$RDFORDD
  T1SV, T1S$RDFVTYP, T1S$RDFLTYP, T1S$RDFLANG, T1S$RDFORDT, T1S$RDFORDN, T1S$RDFORDD
  T1PV, T1P$RDFVTYP, T1P$RDFLTYP, T1P$RDFLANG, T1P$RDFORDT, T1P$RDFORDN, T1P$RDFORDD
  T1OV, T1O$RDFVTYP, T1O$RDFLTYP, T1O$RDFLANG, T1O$RDFORDT, T1O$RDFORDN, T1O$RDFORDD)
```

The same column names for the MVX$SO join view are specified as well.

When a bitmap index is created on a SS-join view, the index is named <MView name><index column name>_I0$. Similarly, the index is named <MView name><index column name>_I1$ for SO-join view. For example, if an index is created on a column T0P in the materialized view MVX, then the index name would be MVXT0P_I0$ for the SS-join view and MVXT0P_I1$ for the SO-join view.

1.18 RDF Support in Oracle SQL Developer

You can use Oracle SQL Developer to perform operations related to the RDF Knowledge Graph feature of Oracle Graph.

For details, see RDF Support in SQL Developer.

1.19 Enhanced RDF ORDER BY Query Processing

Effective with Oracle Database Release 12.2, queries on RDF data that use SPARQL ORDER BY semantics are processed more efficiently than in previous releases.

This internal efficiency involves the use of the ORDER_TYPE, ORDER_NUM, and ORDER_DATE columns in the RDF_VALUE$ metadata table (documented in Statements). The values for these three columns are populated during loading, and this enables ORDER BY queries to reduce internal function calls and to execute faster.

Effective with Oracle Database Release 12.2, the procedure SEM_APIS.ADD_DATATYPE_INDEX creates an index on the ORDER_NUM column for numeric types (xsd:float, xsd:double, and xsd:decimal and all of its subtypes) and an index on ORDER_DATE column for date-related types (xsd:date, xsd:time, and xsd:dateTime) instead of a function-based index as in previous versions. If you want to continue using a function-based index for these data types, you should use the FUNCTION=T option of the SEM_APIS.ADD_DATATYPE_INDEX procedure. For example (assuming a schema-private RDF network named NET1 owned by a database user named RDFUSER):

```
EXECUTE sem_apis.add_datatype_index('http://www.w3.org/2001/
XMLSchema#decimal', options=>'FUNCTION=T', network_owner=>'RDFUSER',
network_name=>'NET1');
```

EXECUTE sem_apis.add_datatype_index('http://www.w3.org/2001/XMLSchema#date', **options=>'FUNCTION=T'**, network_owner=>'RDFUSER', network_name=>'NET1');

1.20 Applying Oracle Machine Learning Algorithms to RDF Data

You can apply Oracle Machine Learning algorithms to RDF data.

Oracle Data Mining requires data to be in a single table or view, and each row represents a single case. Therefore, RDF data needs to be defined as a view mimicking this structure. To accomplish that, do the following:

1. Find the number of predicates of interest: P1, P2, P3, … , Pn.
2. Create a view with columns (S, C1, C2, C3, …. , Cn), where columns correspond to the subject, P1, P2, …, and Pn.

   Depending upon requirements, such as a text column that needs to be defined in a table, you can also create a table.

   Convert numerical values using the TO_NUMBER or CAST function.

   For example:

   ```
   CREATE VIEW ML_TAB (S, C1, C2, C3, … , Cn)
   AS
   SELECT subj, O1, to_number(O2), CAST (O3 AS INTEGER), ... , On
   FROM TABLE(SEM_MATCH(
   'SELECT ?subj ?O1 ?O2 ?O3 … ?On
    WHERE {
      OPTIONAL { ?subj P1 ?O1 }
      OPTIONAL { ?subj P2 ?O2 }
      OPTIONAL { ?subj P3 ?O3 }
   ….
      OPTIONAL { ?subj Pn ?On }
    }'
   , SEM_MODELS('M1')
   ,null, null, null, null));
   ```

Now the view looks something like this:

```
SQL> SELECT * FROM ML_TAB;
S           C1                        C2              C3
---------- -------------------- -------------------- --------------------
S1            O11                O21                O31
S2            O21                                   O32
S3                               O23
```

After you have this structure defined, you can directly apply Oracle Machine Learning algorithms on this view. Oracle Data Mining deals with three types of attributes:

- numerical attribute
- categorical attribute
- unstructured text

You must separate the data into three groups based on the data types of the three types of attributes.

1.21 RDF Graph Management Examples (PL/SQL and Java)

PL/SQL examples are provided in this topic.

For Java examples, see RDF Graph Support for Apache Jena.

- Example: Journal Article Information
- Example: Family Information

1.21.1 Example: Journal Article Information

This section presents a simplified PL/SQL example of RDF graph for statements about journal articles. Example 1-129 contains descriptive comments, refers to concepts that are explained in this chapter, and uses functions and procedures documented in SEM_APIS Package Subprograms.

Example 1-129 Using an RDF Graph for Journal Article Information

```
-- Basic steps:
-- After you have connected as a privileged user and called
-- SEM_APIS.CREATE_RDF_NETWORK to create a schema for storing RDF data,
-- connect as a regular database user and do the following.

-- 1. For each desired network, create an RDF graph(SEM_APIS.CREATE_RDF_GRAPH).
-- Note that we are using the schema-private network NET1 created in
-- "Quick Start for Using RDF Data".

EXECUTE SEM_APIS.CREATE_RDF_GRAPH('articles', 'null', 'null', network_owner=>'RDFUSER',
network_name=>'NET1');

-- Information to be stored about some fictitious articles:
-- Article1, titled "All about XYZ" and written by Jane Smith, refers
--   to Article2 and Article3.
-- Article2, titled "A review of ABC" and written by Joe Bloggs,
--   refers to Article3.
-- Seven SQL statements to store the information. In each statement:
-- Each article is referred to by its complete URI The URIs in
```

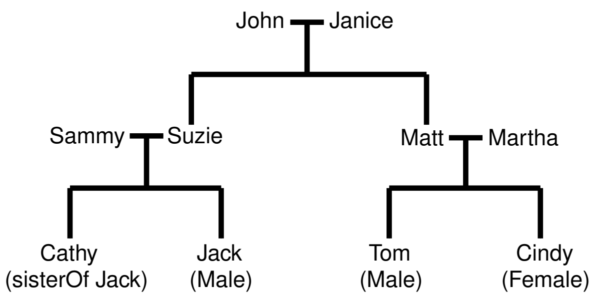
```
--   this example are fictitious.
-- Each property is referred to by the URL for its definition, as
--   created by the Dublin Core Metadata Initiative.

-- 2. Use SEM_APIS.UPDATE_RDF_GRAPH to insert data with SPARQL Update statements

BEGIN
  SEM_APIS.UPDATE_RDF_GRAPH('articles',
   'PREFIX  nature: <http://nature.example.com/>
    PREFIX      dc: <http://purl.org/dc/elements/1.1/>
    PREFIX dcterms: <http://purl.org/dc/terms/>

    INSERT DATA {

      # article1 has the title "All about XYZ".
      # article1 was created (written) by Jane Smith.
      # article1 references (refers to) article2 and article3
      nature:article1 dc:title "All about XYZ" ;
                      dc:creator "Jane Smith" ;
                      dcterms:references nature:article2,
                                         nature:article3 .

      # article2 has the title "A review of ABC".
      # article2 was created (written) by Joe Bloggs.
      # article2 references (refers to) article3.
      nature:article2 dc:title "A Review of ABC" ;
                      dc:creator "Joe Bloggs" ;
                      dcterms:references nature:article3 .
    }',
   network_owner=>'RDFUSER',
   network_name=>'NET1');
END;
/

-- 3. Query RDF data with SEM_MATCH table function.
-- 3.a Get all article authors and titles
SELECT author$rdfterm, title$rdfterm
FROM TABLE(SEM_MATCH(
'PREFIX dc: <http://purl.org/dc/elements/1.1/>
 SELECT ?author ?title
 WHERE { ?article dc:creator ?author
                ; dc:title  ?title . }'
, SEM_MODELS('articles')
, null, null, null, null
, ' PLUS_RDFT=VC '
, null, null
, 'RDFUSER', 'NET1'));

-- 3.b Find all articles referenced by Article1
SELECT ref$rdfterm
FROM TABLE(SEM_MATCH(
'PREFIX dcterms: <http://purl.org/dc/terms/>
 PREFIX  nature: <http://nature.example.com/>
 SELECT ?ref
 WHERE { nature:article1 dcterms:references ?ref . }'
, SEM_MODELS('articles')
, null, null, null, null
, ' PLUS_RDFT=VC '
, null, null
, 'RDFUSER', 'NET1'));
```

1.21.2 Example: Family Information

This section presents a simplified PL/SQL example of an RDF graph for statements about family tree (genealogy) information. Example 1-129 contains descriptive comments, refers to concepts that are explained in this chapter, and uses functions and procedures documented in SEM_APIS Package Subprograms.

The family relationships in this example reflect the family tree shown in Figure 1-3. This figure also shows some of the information directly stated in the example: Cathy is the sister of Jack, Jack and Tom are male, and Cindy is female.

Figure 1-3 Family Tree for RDF Example



Example 1-130 Using an RDF graph for Family Information

```
-- Preparation: create tablespace; enable RDF support.
-- Connect as a privileged user. Example: CONNECT SYSTEM/password-for-SYSTEM
-- Create a tablespace for the RDF data. Example:
CREATE TABLESPACE rdf_tblspace
  DATAFILE 'rdf_tblspace.dat'
    SIZE 128M REUSE
    AUTOEXTEND ON NEXT 128M MAXSIZE 4G
  SEGMENT SPACE MANAGEMENT AUTO;

-- Call SEM_APIS.CREATE_RDF_NETWORK to create a schema-private RDF
-- network named NET1 owned by RDFUSER, which will create database
-- objects to store RDF data. Example:
EXECUTE SEM_APIS.CREATE_RDF_NETWORK('rdf_tblspace', network_owner=>'RDFUSER',
network_name=>'NET1');

-- Connect as the user that is to perform the RDF operations (not SYSTEM),
-- and do the following:
-- 1. For each desired network, create an RDF graph (SEM_APIS.CREATE_RDF_GRAPH).
-- 2. Use various subprograms and constructors.
-- Create the RDF graph.EXECUTE SEM_APIS.CREATE_RDF_GRAPH('family', 'null', 'null',
network_owner=>'RDFUSER', network_name=>'NET1');

-- Insert RDF triples using SEM_APIS.UPDATE_RDF_GRAPH. These express the following
information:
-----------------
-- John and Janice have two children, Suzie and Matt.
-- Matt married Martha, and they have two children:
--   Tom (male) and Cindy (female).
-- Suzie married Sammy, and they have two children:
--   Cathy (female) and Jack (male).

-- Person is a class that has two subslasses: Male and Female.
```

```
-- parentOf is a property that has two subproperties: fatherOf and motherOf.
-- siblingOf is a property that has two subproperties: brotherOf and sisterOf.
-- The domain of the fatherOf and brotherOf properties is Male.
-- The domain of the motherOf and sisterOf properties is Female.
------------------------

BEGIN

  -- Insert some TBox (schema) information.
  SEM_APIS.UPDATE_RDF_GRAPH('family',
   'PREFIX    rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
    PREFIX   rdfs: <http://www.w3.org/2000/01/rdf-schema#>
    PREFIX family: <http://www.example.org/family/>

    INSERT DATA {

      # Person is a class.
      family:Person rdf:type rdfs:Class .

      # Male is a subclass of Person.
      family:Male rdfs:subClassOf family:Person .

      # Female is a subclass of Person.
      family:Female rdfs:subClassOf family:Person .

      # siblingOf is a property.
      family:siblingOf rdf:type rdf:Property .

      # parentOf is a property.
      family:parentOf rdf:type rdf:Property .

      # brotherOf is a subproperty of siblingOf.
      family:brotherOf rdfs:subPropertyOf family:siblingOf .

      # sisterOf is a subproperty of siblingOf.
      family:sisterOf rdfs:subPropertyOf family:siblingOf .

      # A brother is male.
      family:brotherOf rdfs:domain family:Male .

      # A sister is female.
      family:sisterOf rdfs:domain family:Female .

      # fatherOf is a subproperty of parentOf.
      family:fatherOf rdfs:subPropertyOf family:parentOf .

      # motherOf is a subproperty of parentOf.
      family:motherOf rdfs:subPropertyOf family:parentOf .

      # A father is male.
      family:fatherOf rdfs:domain family:Male .

      # A mother is female.
      family:motherOf rdfs:domain family:Female .
    }',
   network_owner=>'RDFUSER',
   network_name=>'NET1');

  -- Insert some ABox (instance) information.
  SEM_APIS.UPDATE_RDF_GRAPH('family',
   'PREFIX    rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
    PREFIX   rdfs: <http://www.w3.org/2000/01/rdf-schema#>
```

```
    PREFIX family: <http://www.example.org/family/>

    INSERT DATA {
      # John is the father of Suzie and Matt
      family:John family:fatherOf family:Suzie .
      family:John family:fatherOf family:Matt .

      # Janice is the mother of Suzie and Matt
      family:Janice family:motherOf family:Suzie .
      family:Janice family:motherOf family:Matt .

      # Sammy is the father of Cathy and Jack
      family:Sammy family:fatherOf family:Cathy .
      family:Sammy family:fatherOf family:Jack .

      # Suzie is the mother of Cathy and Jack
      family:Suzie family:motherOf family:Cathy .
      family:Suzie family:motherOf family:Jack .

      # Matt is the father of Tom and Cindy
      family:Matt family:fatherOf family:Tom .
      family:Matt family:fatherOf family:Cindy .

      # Martha is the mother of Tom and Cindy
      family:Martha family:motherOf family:Tom .
      family:Martha family:motherOf family:Cindy .

      # Cathy is the sister of Jack
      family:Cathy family:sisterOf family:Jack .

      # Jack is male
      family:Jack rdf:type family:Male .

      # Tom is male.
      family:Tom rdf:type family:Male .

      # Cindy is female.
      family:Cindy rdf:type family:Female .
    }',
   network_owner=>'RDFUSER',
   network_name=>'NET1');

END;
/


-- RDFS inferencing in the family RDF graph
BEGIN
  SEM_APIS.CREATE_INFERRED_GRAPH(
    'rdfs_rix_family',
    SEM_Models('family'),
    SEM_Rulebases('RDFS'),
    network_owner=>'RDFUSER',
    network_name=>'NET1');
END;
/

-- Select all males from the family graph, without inferencing.
-- (Returns only Jack and Tom.)
SELECT m$rdfterm
  FROM TABLE(SEM_MATCH(
    'PREFIX  rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
```

```
     PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
     PREFIX     : <http://www.example.org/family/>
     SELECT ?m
     WHERE {?m rdf:type :Male}',
    SEM_Models('family'),
    null,  null,  null, null,
    ' PLUS_RDFT=VC ',
    null, null,
    'RDFUSER', 'NET1'));

-- Select all males from the family graph, with RDFS inferencing.
-- (Returns Jack, Tom, John, Sammy, and Matt.)
SELECT m$rdfterm
  FROM TABLE(SEM_MATCH(
    'PREFIX  rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
     PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
     PREFIX     : <http://www.example.org/family/>
     SELECT ?m
     WHERE {?m rdf:type :Male}',
    SEM_Models('family'),
    SEM_Rulebases('RDFS'),
    null, null, null,
    ' PLUS_RDFT=VC ',
    null, null,
    'RDFUSER', 'NET1'));

-- General inferencing in the family graph

EXECUTE SEM_APIS.CREATE_RULEBASE('family_rb', network_owner=>'RDFUSER',
network_name=>'NET1');

INSERT INTO rdfuser.net1#semr_family_rb VALUES(
  'grandparent_rule',
  '(?x :parentOf ?y) (?y :parentOf ?z)',
  NULL,
  '(?x :grandParentOf ?z)',
  SEM_ALIASES(SEM_ALIAS('','http://www.example.org/family/')));

COMMIT;

-- Because a new rulebase has been created, and it will be used in the
-- inferred graph, drop the preceding inferred graph and then re-create it.
EXECUTE SEM_APIS.DROP_INFERRED_GRAPH ('rdfs_rix_family', network_owner=>'RDFUSER',
network_name=>'NET1');

-- Re-create the inferred graph.
BEGIN
  SEM_APIS.CREATE_INFERRED_GRAPH(
    'rdfs_rix_family',
    SEM_Models('family'),
    SEM_Rulebases('RDFS','family_rb'),
    network_owner=>'RDFUSER', network_name=>'NET1');
END;
/

-- Select all grandfathers and their grandchildren from the family graph,
-- without inferencing. (With no inferencing, no results are returned.)
SELECT x$rdfterm grandfather, y$rdfterm grandchild
  FROM TABLE(SEM_MATCH(
    'PREFIX  rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
     PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
     PREFIX     : <http://www.example.org/family/>
```

```
     SELECT ?x ?y
     WHERE {?x :grandParentOf ?y . ?x rdf:type :Male}',
    SEM_Models('family'),
    null, null,  null, null,
    ' PLUS_RDFT=VC ',
    null, null,
    'RDFUSER', 'NET1'));

-- Select all grandfathers and their grandchildren from the family graph.
-- Use inferencing from both the RDFS and family_rb rulebases.
SELECT x$rdfterm grandfather, y$rdfterm grandchild
  FROM TABLE(SEM_MATCH(
    'PREFIX  rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
     PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
     PREFIX     : <http://www.example.org/family/>
     SELECT ?x ?y
     WHERE {?x :grandParentOf ?y . ?x rdf:type :Male}',
    SEM_Models('family'),
    SEM_Rulebases('RDFS','family_rb'),
    null,  null, null,
    ' PLUS_RDFT=VC ',
    null, null,
    'RDFUSER', 'NET1'));
```

1.22 Software Naming Changes Since Release 11.1

Because the support for RDF data has been expanded beyond the original focus on RDF, the names of many software objects (PL/SQL packages, functions and procedures, system tables and views, and so on) have been changed as of Oracle Database Release 11.1.

In most cases, the change is to replace the string *RDF* with *SEM*. although in some cases it may be to replace *SDO_RDF* with *SEM*.

All valid code that used the pre-Release 11.1 names will continue to work; your existing applications will not be broken. However, it is suggested that you change old applications to use new object names, and you should use the new names for any new applications. This manual will document only the new names.

Table 1-32 lists the old and new names for some objects related to support for semantic technologies, in alphabetical order by old name.

Table 1-32 Semantic Technology Software Objects: Old and New Names

Old Name	New Name
RDF_ALIAS data type	SEM_ALIAS
RDF_MODEL$ view	SEM_MODEL$
RDF_RULEBASE_INFO view	SEM_RULEBASE_INFO
RDF_RULES_INDEX_DATASETS view	SEM_RULES_INDEX_DATASETS
RDF_RULES_INDEX_INFO view	SEM_RULES_INDEX_INFO
RDFI_*rules-index-name* view	SEMI_*rules-index-name*
RDFM_*model-name* view	SEMM_*model-name*
RDFR_*rulebase-name* view	SEMR_*rulebase-name*
SDO_RDF package	SEM_APIS
SDO_RDF_INFERENCE package	SEM_APIS

Table 1-32 (Cont.) Semantic Technology Software Objects: Old and New Names

Old Name	New Name
SDO_RDF_MATCH table function	SEM_MATCH
SDO_RDF_MODELS data type	SEM_MODELS
SDO_RDF_RULEBASES data type	SEM_RULEBASES

1.23 For More Information About RDF Graph

More information is available about RDF graph support and related topics.

See the following resources:

- Oracle Graph RDF Graph page (OTN), which includes links for downloads, technical and business white papers, a discussion forum, and other sources of information: `http://www.oracle.com/technetwork/database/options/spatialandgraph/overview/rdfsemantic-graph-1902016.html`
- World Wide Web Consortium (W3C) *RDF Primer*: `http://www.w3.org/TR/rdf-primer/`
- World Wide Web Consortium (W3C) *OWL Web Ontology Language Reference*: `http://www.w3.org/TR/owl-ref/`

1.24 Required Migration of Pre-12.2 RDF Data

If you have any RDF data created using Oracle Database 11.1. 11.2, or 12.1, then before you use it in an Oracle Database 12.2 environment, you must migrate this data.

To perform the migration, use the SEM_APIS.MIGRATE_DATA_TO_CURRENT procedure. This applies not only to your existing RDF data, but also to any other RDF data introduced into your environment if that data was created using Oracle Database 11.1. 11.2, or 12.1

The reason for this requirement is for optimal performance of queries that use ORDER BY. Effective with Release 12.2, Oracle Database creates, populates, and uses the ORDER_TYPE, ORDER_NUM, and ORDER_DATE columns (new in Release 12.2) in the RDF_VALUE$ table (described in Statements). The SEM_APIS.MIGRATE_DATA_TO_CURRENT procedure populates these order-related columns. If you do not do this, those columns will be null for existing data.

You run this procedure after upgrading to Oracle Database Release 12.2. If you later bring into your Release 12.2 environment any RDF data that was created using an earlier release, you must also run the procedure before using that data. Running the procedure can take a long time with large amounts of RDF data, so consider that in deciding when to tun it. (Note that using the `INS_AS_SEL=T` option improves the performance of the SEM_APIS.MIGRATE_DATA_TO_CURRENT procedure with large data sets.)

1.25 Oracle RDF Graph Features that Support Accessibility

This section describes the accessibility support provided by Oracle RDF Graph features.

- The Oracle Adapter for Eclipse RDF4J enables developers to build applications that can interact with the RDF graph feature in Oracle Database using the Eclipse RDF4J framework. See the WCAG Documentation to create applications based on WCAG 2.1 accessibility standards.

- The RDF Query UI is based on Oracle JET. For more information about accessibility of Oracle JET components, see the Oracle JET Documentation.
- Additionally, by enabling accessibility in RDF Query UI, all SPARQL query execution results are displayed in tabular format. See the Accessibility section for more information.

2 Quick Start for Using RDF Data

This section provides the steps to help you get started on working with RDF data in an Oracle Database.

To work with RDF data, you must create an RDF network in the user schema. Follow these general steps as applicable to your environment.

- Getting Started with RDF Data in a Schema-Private Network
- Quick Start for Using RDF Data in Oracle Autonomous Database

2.1 Getting Started with RDF Data in a Schema-Private Network

1. Create a tablespace for the system tables. You must be connected as a user with appropriate privileges to create the tablespace. The following example creates a tablespace named `rdf_tblspace`:

   ```
   CREATE TABLESPACE rdf_tblspace
    DATAFILE 'rdf_tblspace.dat' SIZE 1024M REUSE
    AUTOEXTEND ON NEXT 256M MAXSIZE UNLIMITED
    SEGMENT SPACE MANAGEMENT AUTO;
   ```

2. Create a database user to work with RDF data in the database and grant the necessary privileges to the database user. You must be connected as a user with appropriate privileges to create the database user.

   The following example creates a network owner user `rdfuser` and grants the necessary privileges to `rdfuser`:

   ```
   CREATE USER rdfuser
   IDENTIFIED BY <password-for-rdfuser>
   QUOTA 5G ON rdf_tblspace;

   GRANT CONNECT, RESOURCE, CREATE VIEW TO rdfuser;
   ```

3. Connect as the network owner user.

   ```
   CONNECT rdfuser/<password-for-rdfuser>
   ```

4. Create a schema-private RDF network.

   Creating an RDF network adds RDF data support to an Oracle database. You must create an RDF network as the intended owner of the schema-private network, specifying a valid tablespace with adequate space.

   The following example creates a schema-private RDF network named `net1` owned by a database user named `rdfuser` using a tablespace named `rdf_tblspace`:

   ```
   EXECUTE SEM_APIS.CREATE_RDF_NETWORK('rdf_tblspace',
   network_owner=>'rdfuser', network_name=>'net1');
   ```

5. Create an RDF graph.

   When you create an RDF graph, you specify the RDF graph name, the table to hold references to RDF data for the graph, and the column of type SDO_RDF_TRIPLE_S in that table.

   The following command creates an RDF graph named `articles` in the `net1` schema-private network.

   ```
   EXECUTE SEM_APIS.CREATE_RDF_GRAPH('articles', NULL, NULL,
   network_owner=>'rdfuser', network_name=>'net1');
   ```

After you create the RDF graph, you can insert triples into the graph, as shown in the examples in RDF Graph Management Examples (PL/SQL and Java).

2.2 Quick Start for Using RDF Data in Oracle Autonomous Database

You can use any of the following options to work with RDF data in Autonomous Database:

- RDF feature of Oracle Graph is included in all versions of Oracle Autonomous Data Warehouse and Oracle Autonomous Transaction Processing in both shared and dedicated deployments.
- RDF Graph Server and Query UI is supported with all versions of Oracle Autonomous Data Warehouse and Oracle Autonomous Transaction Processing in both shared and dedicated deployments. RDF Graph Server enables you to create a SPARQL endpoint and perform other RDF graph data management operations using the Query UI.
- Graph Studio, a component of Autonomous Database in shared deployments, allows you to easily create, manage, query, analyze, and visualize RDF graphs. This web-based graph interface is supported on both Data Warehouse and Transaction Processing workload types.
- Getting Started with RDF Data in Oracle Autonomous Database
  This tutorial describes how you can quickly get started with RDF data in Autonomous Database.
- Deploying RDF Graph Server and Query UI from Oracle Cloud Marketplace

2.2.1 Getting Started with RDF Data in Oracle Autonomous Database

This tutorial describes how you can quickly get started with RDF data in Autonomous Database.

You can run the SQL statements shown in the steps through one of the following options:

- Using any of the SQL based Oracle Database tools that is connected with your Autonomous Database. See Connect to Autonomous Database Using Oracle Database Tools for more details.
- Using the built-in Database Actions which provides a web-based interface. See Connect with Built-in Oracle Database Actions for more details.

1. Connect to your autonomous database instance as a user with administrative privileges and create a network owner user.

```
CREATE USER rdfuser
IDENTIFIED BY <password-for-rdfuser>
QUOTA 5G ON DATA;
```

> **Note:**
>
> If you are using Database Actions, then you must **REST Enable** the user in order to enable the new user to access Database Actions. See Create User for more details.

2. Grant the required privileges to the newly created network owner user.

   You must be connected as a user with administrative privileges to run the following statement:

```
GRANT CONNECT, RESOURCE, CREATE VIEW TO rdfuser;
```

> **Note:**
>
> If you are using Database Actions to create the new user in the preceding step, then the `CONNECT` and the `RESOURCE` privileges are provided by default. Hence, you must only grant the `CREATE VIEW` privilege to the new user.

3. Connect as the network owner user.

```
CONNECT rdfuser/<password-for-rdfuser>
```

4. Create an RDF network by calling `SEM_APIS.CREATE_RDF_NETWORK`.

   You must create an RDF network as the intended owner of the schema-private network on the tablespace `DATA`.

   The following example creates a schema-private RDF network named `net1` owned by network owner user `rdfuser` using the `DATA` tablespace.

```
EXECUTE SEM_APIS.CREATE_RDF_NETWORK('DATA', network_owner=>'rdfuser',
network_name=>'net1');
```

5. Create an RDF graph by calling `SEM_APIS.CREATE_RDF_GRAPH`.

   The following example creates an RDF graph named `articles` in the `net1` schema-private network.

```
EXECUTE SEM_APIS.CREATE_RDF_GRAPH('articles', NULL, NULL,
network_owner=>'rdfuser', network_name=>'net1');
```

6. Insert triples into the RDF graph using one of the following options.

   - Use the **SEM_APIS.UPDATE_RDF_GRAPH** procedure to insert data:

> **Note:**
>
> The SEM_APIS.UPDATE_RDF_GRAPH function is supported only if Oracle JVM is enabled on your Oracle Autonomous Database Serverless deployments. To enable Oracle JVM, see Use Oracle Java in *Using Oracle Autonomous Database Serverless* for more information.

You can insert triples into an RDF graph using SEM_APIS.UPDATE_RDF_GRAPH as shown in the examples in RDF Graph Management Examples (PL/SQL and Java).

- Use the **SQL** `INSERT` statement to insert data:
  For example:

```
INSERT INTO rdfuser.net1#rdft_articles(triple) VALUES (
  SDO_RDF_TRIPLE_S ('articles','<http://nature.example.com/Article1>',
    '<http://purl.org/dc/elements/1.1/title>','"All about XYZ"',
    network_owner=>'RDFUSER', network_name=>'NET1'));

INSERT INTO rdfuser.net1#rdft_articles(triple) VALUES (
  SDO_RDF_TRIPLE_S ('articles','<http://nature.example.com/Article1>',
    '<http://purl.org/dc/elements/1.1/creator>','"Jane Smith"',
    network_owner=>'RDFUSER', network_name=>'NET1'));

INSERT INTO rdfuser.net1#rdft_articles(triple) VALUES (
  SDO_RDF_TRIPLE_S ('articles',
    '<http://nature.example.com/Article1>',
    '<http://purl.org/dc/terms/references>',
    '<http://nature.example.com/Article2>',
    network_owner=>'RDFUSER', network_name=>'NET1'));

INSERT INTO rdfuser.net1#rdft_articles(triple) VALUES (
  SDO_RDF_TRIPLE_S ('articles','<http://nature.example.com/Article2>',
    '<http://purl.org/dc/elements/1.1/title>','"A review of ABC"',
    network_owner=>'RDFUSER', network_name=>'NET1'));

INSERT INTO rdfuser.net1#rdft_articles(triple) VALUES (
  SDO_RDF_TRIPLE_S ('articles','<http://nature.example.com/Article2>',
    '<http://purl.org/dc/elements/1.1/creator>','"Joe Bloggs"',
    network_owner=>'RDFUSER', network_name=>'NET1'));

INSERT INTO rdfuser.net1#rdft_articles(triple) VALUES (
  SDO_RDF_TRIPLE_S ('articles',
    '<http://nature.example.com/Article2>',
    '<http://purl.org/dc/terms/references>',
    '<http://nature.example.com/Article3>',
    network_owner=>'RDFUSER', network_name=>'NET1'));
```

7. Execute SPARQL queries on the inserted data using one of the following options as applicable.
   - Query RDF data with the **SEM_MATCH** table function:

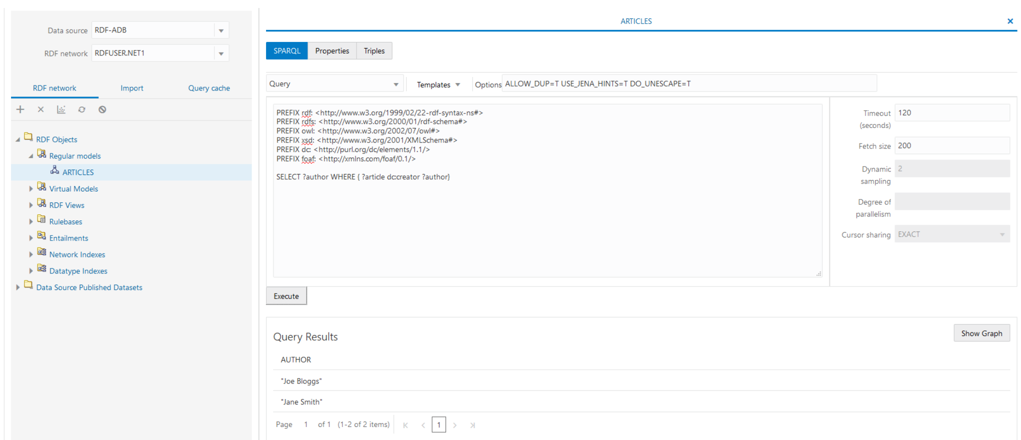
> **Note:**
>
> The SEM_MATCH table function is supported only if Oracle JVM is enabled on your Oracle Autonomous Database Serverless deployments. To enable Oracle JVM, see Use Oracle Java in *Using Oracle Autonomous Database Serverless* for more information.

You can query the inserted triples data using the SEM_MATCH table function as shown in the examples in RDF Graph Management Examples (PL/SQL and Java).

- Query RDF data using RDF Graph Server and Query UI:
  See Deploying RDF Graph Server and Query UI from Oracle Cloud Marketplace to launch the RDF Query UI application.

  You can query the inserted data by running SPARQL queries on the SPARQL query page in RDF Graph Query UI.

Figure 2-1 Running SPARQL Query in RDF Graph Query UI



- Query using SPARQL editor in SQL Developer:
  You can also run SPARQL queries using SPARQL editor in SQL Developer.

> **Note:**
>
> SQL Developer versions earlier than 21.2 is supported only if Oracle JVM is enabled on your Oracle Autonomous Database Serverless deployments. To enable Oracle JVM, see Use Oracle Java in *Using Oracle Autonomous Database Serverless* for more information.

See Connect Oracle SQL Developer with a Wallet for creating a connection to your autonomous database instance using Cloud Wallet.

2.2.2 Deploying RDF Graph Server and Query UI from Oracle Cloud Marketplace

You can set up RDF Graph Server and Query UI in your Autonomous Database instance using the Oracle Cloud Marketplace image.

As a prerequisite, you must have the following already created:

- Oracle Autonomous Database (shared or dedicated infrastructure) created using your Oracle Cloud account
- Virtual Cloud Network (VCN) in your tenancy
- OCI compartment to create the stack instance
- SSH Key pair for `ssh` access to the instance

The Oracle Cloud Infrastructure (OCI) Marketplace displays two listings for Oracle RDF Graph Server and Query UI. However, the deployment varies depending on the pricing model as shown:

- **Free:** Apache Tomcat Server deployment
- **BYOL:** Oracle WebLogic Server deployment

The following steps apply to both Autonomous Data Warehouse or Autonomous Transaction Processing workload types as applicable to your Autonomous Database.

1. Sign in to the OCI console and navigate to Marketplace.
2. Search **RDF** on the Cloud Marketplace page and click the **RDF Graph Server and Query UI** listing that applies to you.
3. Review, accept the Oracle Standard Terms and Restrictions and click **Launch Stack**.

   The Stack setup wizard gets triggered.
4. Enter the appropriate metadata, selecting the required options to create the **Compute Instance** and configure the **Instance Network** variables.
5. Enter the ADMIN user credentials for your application server under **Advanced Configuration**.
6. Review the information and click **Create**.

   The stack deployment gets invoked and you can monitor the job progress on the Job Details page.
   Once the job completes and the stack is created successfully, the status shows as **SUCCEEDED** on the Job Details page.

   The RDF Graph Server and Query UI instance is now provisioned.
7. Scroll down to the bottom of the logs section and note the public URL to launch RDF Graph Server and Query UI.

   The URL follows the format as shown:

   - **Apache Tomcat Deployment:** `https://<public_IP>:4040/orardf`
   - **WebLogic Server Deployment:** `https://<public_IP>:8001/orardf`
8. Launch the RDF Graph Server and Query UI application in your browser.

   The **RDF Graph** login screen appears. See RDF Graph Server and Query UI for more details.

3
OWL Concepts

You should understand key concepts related to the support for a subset of the Web Ontology Language (OWL).

This chapter builds on the information in RDF Graph Overview, and it assumes that you are familiar with the major concepts associated with OWL, such as ontologies, properties, and relationships. For detailed information about OWL, see the *OWL Web Ontology Language Reference* at `http://www.w3.org/TR/owl-ref/`.

- Ontologies
  An **ontology** is a shared conceptualization of knowledge in a particular domain.
- Using OWL Inferencing
  You can use inference rules to perform native OWL inferencing.
- Using Semantic Operators to Query Relational Data
  You can use semantic operators to query relational data in an ontology-assisted manner, based on the semantic relationship between the data in a table column and terms in an ontology.

3.1 Ontologies

An **ontology** is a shared conceptualization of knowledge in a particular domain.

It consists of a collection of classes, properties, and optionally instances. Classes are typically related by class hierarchy (subclass/ superclass relationship). Similarly, the properties can be related by property hierarchy (subproperty/ superproperty relationship). Properties can be symmetric or transitive, or both. Properties can also have domain, ranges, and cardinality constraints specified for them.

RDFS-based ontologies only allow specification of class hierarchies, property hierarchies, `instanceOf` relationships, and a domain and a range for properties.

OWL ontologies build on RDFS-based ontologies by additionally allowing specification of property characteristics. OWL ontologies can be further classified as OWL-Lite, OWL-DL, and OWL Full. OWL-Lite restricts the cardinality minimum and maximum values to 0 or 1. OWL-DL relaxes this restriction by allowing minimum and maximum values. OWL Full allows instances to be also defined as a class, which is not allowed in OWL-DL and OWL-Lite ontologies.

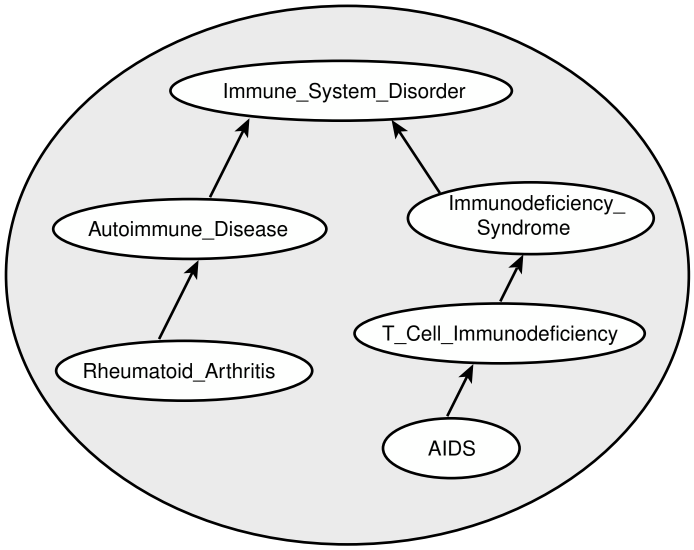
Supported OWL Subsets describes OWL capabilities that are supported and not supported with RDF data.

- Example: Disease Ontology
- Supported OWL Subsets

3.1.1 Example: Disease Ontology

Figure 3-1 shows part of a disease ontology, which describes the classes and properties related to certain diseases. One requirement is to have a PATIENTS data table with a column named DIAGNOSIS, which must contain a value from the `Diseases_and_Disorders` class hierarchy.

Figure 3-1 Disease Ontology Example



In the disease ontology shown in Figure 3-1, the diagnosis `Immune_System_Disorder` includes two subclasses, `Autoimmune_Disease` and `Immunodeficiency_Syndrome`. The `Autoimmune_Disease` diagnosis includes the subclass `Rheumatoid_Arthritis`; and the `Immunodeficiency_Syndrome` diagnosis includes the subclass `T_Cell_Immunodeficiency`, which includes the subclass `AIDS`.

The data in the PATIENTS table might include the PATIENT_ID and DIAGNOSIS column values shown in Table 3-1.

Table 3-1 PATIENTS Table Example Data

PATIENT_ID	DIAGNOSIS
1234	Rheumatoid_Arthritis
2345	Immunodeficiency_Syndrome
3456	AIDS

To query ontologies, you can use the SEM_MATCH table function or the SEM_RELATED operator and its ancillary operators.

Related Topics

- Using the SEM_MATCH Table Function to Query RDF Data
  To query RDF data, use the SEM_MATCH table function.
- Using Semantic Operators to Query Relational Data
  You can use semantic operators to query relational data in an ontology-assisted manner, based on the semantic relationship between the data in a table column and terms in an ontology.

3.1.2 Supported OWL Subsets

This section describes OWL vocabulary subsets that are supported.

Oracle Database supports the RDFS++, OWLSIF, and OWLPrime vocabularies, which have increasing expressivity, as well as OWL 2 RL. Each supported vocabulary has a corresponding rulebase; however, these rulebases do not need to be populated because the underlying inference rules of these three vocabularies are internally implemented. The supported vocabularies are as follows:

- RDFS++: A minimal extension to RDFS; which is RDFS plus `owl:sameAs` and `owl:InverseFunctionalProperty`.
- OWLSIF: OWL with IF Semantic, with the vocabulary and semantics proposed for pD* semantics in *Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary*, by H.J. Horst, Journal of Web Semantics 3, 2 (2005), 79–115.
- OWLPrime: The following OWL capabilities:
  - Basics: class, subclass, property, subproperty, domain, range, type
  - Property characteristics: transitive, symmetric, functional, inverse functional, inverse
  - Class comparisons: equivalence, disjointness
  - Property comparisons: equivalence
  - Individual comparisons: same, different
  - Class expressions: complement
  - Property restrictions: `hasValue`, `someValuesFrom`, `allValuesFrom`

    As with pD*, the supported semantics for these value restrictions are only intensional (IF semantics).
- OWL 2 RL: Described in the "OWL 2 RL" section of the W3C *OWL 2 Web Ontology Language Profiles* recommendation (`http://www.w3.org/TR/owl2-profiles/#OWL_2_RL`) as: "The OWL 2 RL profile is aimed at applications that require scalable reasoning without sacrificing too much expressive power. It is designed to accommodate both OWL 2 applications that can trade the full expressivity of the language for efficiency, and RDF(S) applications that need some added expressivity from OWL 2."

  The system-defined rulebase `OWL2RL` supports all the standard production rules defined for OWL 2 RL. As with `OWLPRIME`, users will not see any rules in this OWL2RL rulebase. The rulebase `OWL2RL` will be created automatically if it does not already exist.

  The following code excerpt uses the OWL2RL rulebase:

  ```
  CREATE TABLE m1_tpl (triple SDO_RDF_TRIPLE_S) COMPRESS;
  EXECUTE
  sem_apis.create_rdf_graph('m1','m1_tpl','triple',network_owner=>'RDFUSER',network_nam
  e=>'NET1');
  -- Insert data into RDF graph M1. Details omitted
  ...
  -- Now run inference using the OWL2RL rulebase
  EXECUTE
  sem_apis.create_inferred_graph('m1_inf',sem_models('m1'),sem_rulebases('owl2rl'),netw
  ork_owner=>'RDFUSER',network_name=>'NET1');
  ```

  Note that inference-related optimizations, such as parallel inference and RAW8, are all applicable when the `OWL2RL` rulebase is used.
- OWL 2 EL: Described in the "OWL 2 EL" section of the W3C *OWL 2 Web Ontology Language Profiles* recommendation (`http://www.w3.org/TR/owl2-profiles/#OWL_2_EL`) as: "The OWL 2 EL profile is designed as a subset of OWL 2 that

– is particularly suitable for applications employing ontologies that define very large numbers of classes and/or properties,
– captures the expressive power used by many such ontologies, and
– for which ontology consistency, class expression subsumption, and instance checking can be decided in polynomial time."

A prime example of OWL 2 EL ontology is the biomedical ontology SNOMED Clinical Terms (SNOMED CT). For information about SNOMED CT, see: `http://www.ihtsdo.org/snomed-ct/`

The system-defined rulebase `OWL2EL` supports the EL syntax.

As with `OWLPRIME` and `OWL2RL`, users will not see any rules in this `OWL2EL` rulebase, and the `OWL2EL` rulebase will be created automatically if it does not already exist.

The following code excerpt uses the `OWL2EL` rulebase against the well known SNOMED ontology:

```
CREATE TABLE snomed_tpl (triple SDO_RDF_TRIPLE_S) COMPRESS;
EXECUTE
sem_apis.create_rdf_graph('snomed','snomed_tpl','triple',network_owner=>'RDFUSER',net
work_name=>'NET1') compress;
-- Insert data into RDF graph SNOMED. Details omitted
...
-- Now run inference using the OWL2EL rulebase
EXECUTE
sem_apis.create_inferred_graph('snomed_inf',sem_models('snomed'),sem_rulebases('owl2e
l'),network_owner=>'RDFUSER',network_name=>'NET1');
```

Note that the `OWL2EL` rulebase support does not include reflexive object properties (`ReflexiveObjectProperty`) simply because a reflexive object property will link every individual with itself, which would probably cause an unnecessary and costly expansion of the inference graph.

Table 3-2 lists the RDFS/OWL vocabulary constructs included in each supported rulebase.

Table 3-2 RDFS/OWL Vocabulary Constructs Included in Each Supported Rulebase

Rulebase Name	RDFS/OWL Constructs Included
RDFS++	all RDFS vocabulary constructs owl:InverseFunctionalProperty owl:sameAs
OWLSIF	all RDFS vocabulary constructs owl:FunctionalProperty owl:InverseFunctionalProperty owl:SymmetricProperty owl:TransitiveProperty owl:sameAs owl:inverseOf owl:equivalentClass owl:equivalentProperty owl:hasValue owl:someValuesFrom owl:allValuesFrom

Table 3-2 (Cont.) RDFS/OWL Vocabulary Constructs Included in Each Supported Rulebase

Rulebase Name	RDFS/OWL Constructs Included
OWLPrime	rdfs:subClassOf rdfs:subPropertyOf rdfs:domain rdfs:range owl:FunctionalProperty owl:InverseFunctionalProperty owl:SymmetricProperty owl:TransitiveProperty owl:sameAs owl:inverseOf owl:equivalentClass owl:equivalentProperty owl:hasValue owl:someValuesFrom owl:allValuesFrom owl:differentFrom owl:disjointWith owl:complementOf
OWL2RL	(As described in `http://www.w3.org/TR/owl2-profiles/#OWL_2_RL`)
OWL2EL	(As described in `http://www.w3.org/TR/owl2-profiles/#OWL_2_EL`)

3.2 Using OWL Inferencing

You can use inference rules to perform native OWL inferencing.

This section creates a simple ontology, performs native inferencing, and illustrates some more advanced features.

- Creating a Simple OWL Ontology
- Performing Native OWL Inferencing
- Performing OWL and User-Defined Rules Inferencing
- Generating OWL Inferencing Proofs
- Validating OWL RDF Graphs and Inferred Graphs
- Using SEM_APIS.CREATE_INFERRED_GRAPH for RDFS Inference
- Enhancing Inference Performance
- Optimizing owl:sameAs Inference
- Performing Incremental Inference
- Using Parallel Inference
- Using Named Graph Based Inferencing (Global and Local)
- Performing Selective Inferencing (Advanced Information)

3.2.1 Creating a Simple OWL Ontology

Example 3-1 creates a simple OWL ontology, inserts one statement that two URIs refer to the same entity, and performs a query using the SEM_MATCH table function.

Example 3-1 Creating a Simple OWL Ontology

```
SQL> CREATE TABLE owltst(id number, triple sdo_rdf_triple_s);
Table created.

SQL> EXECUTE
sem_apis.create_rdf_graph('owltst','owltst','triple',network_owner=>'RDFUSER',network_nam
e=>'NET1');
PL/SQL procedure successfully completed.

SQL> INSERT INTO owltst VALUES (1, sdo_rdf_triple_s('owltst',
       'http://example.com/name/John', 'http://www.w3.org/2002/07/owl#sameAs',
       'http://example.com/name/JohnQ','RDFUSER','NET1'));
1 row created.

SQL> commit;

SQL> -- Use SEM_MATCH to perform a simple query.
SQL> select s$rdfterm,p$rdfterm,o$rdfterm from table(SEM_MATCH('SELECT * WHERE {?s ?p  ?
o}', SEM_Models('OWLTST'),
           null,  null, null, null, 'PLUS_RDFT=VC', null, null, 'RDFUSER', 'NET1'));
```

3.2.2 Performing Native OWL Inferencing

Example 3-2 calls the SEM_APIS.CREATE_INFERRED_GRAPH procedure. You do not need to create the rulebase and add rules to it, because the OWL rules are already built into the RDF Graph inferencing engine.

Example 3-2 Performing Native OWL Inferencing

```
SQL> -- Invoke the following command to run native OWL inferencing that
SQL> -- understands the vocabulary defined in the preceding section.
SQL>
SQL>  EXECUTE sem_apis.create_inferred_graph('owltst_idx', sem_models('owltst'),
sem_rulebases('OWLPRIME'), network_owner=>'RDFUSER', network_name=>'NET1');
PL/SQL procedure successfully completed.

SQL> -- The following view is generated to represent the inferred graph (rules index).
SQL> desc RDFUSER.NET1#semi_owltst_idx;

SQL> -- Run the preceding query with an additional rulebase parameter to list
SQL> -- the original graph plus the inferred triples.
SQL> SELECT s$rdfterm,p$rdfterm,o$rdfterm FROM table(SEM_MATCH('SELECT * WHERE {?s ?p  ?
o}', SEM_MODELS('OWLTST'),
           SEM_RULEBASES('OWLPRIME'),  null, null, null, null, 'PLUS_RDFT=VC', null,
null, 'RDFUSER', 'NET1'));
```

3.2.3 Performing OWL and User-Defined Rules Inferencing

Example 3-3 creates a user-defined rulebase, inserts a simplified `uncleOf` rule (stating that the brother of one's father is one's uncle), and calls the SEM_APIS.CREATE_INFERRED_GRAPH procedure.

Example 3-3 Performing OWL and User-Defined Rules Inferencing

```
SQL> -- First, insert the following assertions.

SQL> INSERT INTO owltst VALUES (1, sdo_rdf_triple_s('owltst',
       'http://example.com/name/John', 'http://example.com/rel/fatherOf',
       'http://example.com/name/Mary', 'RDFUSER', 'NET1'));


SQL> INSERT INTO owltst VALUES (1, sdo_rdf_triple_s('owltst',
       'http://example.com/name/Jack', 'http://example.com/rel/brotherOf',
       'http://example.com/name/John', 'RDFUSER', 'NET1'));

SQL> -- Create a user-defined rulebase.

SQL> EXECUTE sem_apis.create_rulebase('user_rulebase', network_owner=>'RDFUSER',
network_name=>'NET1');

SQL> -- Insert a simple "uncle" rule.

SQL> INSERT INTO RDFUSER.NET1#SEMR_USER_RULEBASE VALUES ('uncle_rule',
'(?x <http://example.com/rel/brotherOf> ?y)(?y <http://example.com/rel/fatherOf> ?z)',
NULL, '(?x <http://example.com/rel/uncleOf> ?z)', null);

SQL>  -- In the following statement, 'USER_RULES=T' is required, to
SQL> --  include the original graph plus the inferred triples.
SQL> EXECUTE sem_apis.create_inferred_graph('owltst2_idx', sem_models('owltst'),
          sem_rulebases('OWLPRIME','USER_RULEBASE'),
          SEM_APIS.REACH_CLOSURE, null, 'USER_RULES=T', network_owner=>'RDFUSER',
network_name=>'NET1');

SQL> -- In the result of the following query, :Jack :uncleOf :Mary is inferred.
SQL> SELECT s$rdfterm,p$rdfterm,o$rdfterm FROM table(SEM_MATCH('SELECT * WHERE {?s ?p  ?
o}',
           SEM_MODELS('OWLTST'),
           SEM_RULEBASES('OWLPRIME','USER_RULEBASE'),  null, null, null, null,
'PLUS_RDFT=VC', null, null, 'RDFUSER', 'NET1'));
```

For performance, the inference engine by default executes each user rule without checking the syntax legality of inferred triples (for example, literal value as a subject, blank node as a predicate) until after the last round of inference. After completing the last inference round, the inference engine removes all syntactically illegal triples without throwing any errors for these triples. However, because triples with illegal syntax may exist during multiple rounds of inference, rules can use these triples as part of their antecedents. For example, consider the following user-defined rules:

- Rule 1:

```
(?s :account ?y)
(?s :country :Spain) --> (?y rdf:type :SpanishAccount)
```

- Rule 2:

```
(?s :account ?y)
(?y rdf:type :SpanishAccount) --> (?s :language "es_ES")
```

Rule 1 finds all Spanish users and designates their accounts as Spanish accounts. Rule 2 sets the language for all users with Spanish accounts to `es_ES` (Spanish). Consider the following data, displayed in Turtle format:

```
:Juan       :account "123ABC4Z"
            :country :Spain
```

```
:Alejandro :account "5678DEF9Y"
           :country :Spain
```

Applying Rule 1 and Rule 2 produces the following inferred triples:

```
(:Juan      :language "es_ES")
(:Alejandro :language "es_ES")
```

Note there are no triples specifying which accounts are of type `:SpanishAccount`. The user-defined rules infer those triples during creation of the inferred graph, but the inference engine removes them after the last round of inference because they contain illegal syntax. The accounts are the literal values, which cannot be used as subjects in an RDF triple.

To force the checking of syntax legality of inferred triples, add the `/*+ ENABLE_SYNTAX_CHECKING */` optimizer hint to the beginning of the rule's FILTER expression. Forcing syntax checking for a rule can result in a performance penalty and will throw an exception for any syntactically illegal triples. The following example, similar to Rule 1, forces syntax checking. (In addition, merely to illustrate the use of a filter expression, the example restricts accounts to those that do not end with the letter '`Z`'.)

```
INSERT INTO RDFUSER.NET1#SEMR_USER_RULEBASE VALUES (
  'spanish_account_rule',
  '(?s <http://example.com/account> ?y)(?y <http://example.com/account> <http://
example.com/Spain>)',
  '/*+ ENABLE_SYNTAX_CHECKING */ y not like ''%Z'' ',
  '(?y <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://example.com/
SpanishAccount>)',
  NULL
);
```

3.2.4 Generating OWL Inferencing Proofs

OWL inference can be complex, depending on the size of the ontology, the actual vocabulary (set of language constructs) used, and the interactions among those language constructs. To enable you to find out how a triple is derived, you can use proof generation during inference. (Proof generation does require additional CPU time and disk resources.)

To generate the information required for proof, specify `PROOF=T` in the call to the SEM_APIS.CREATE_INFERRED_GRAPH procedure, as shown in the following example:

```
EXECUTE sem_apis.create_inferred_graph('owltst_idx', sem_models('owltst'), -
  sem_rulebases('owlprime'), SEM_APIS.REACH_CLOSURE, 'SAM', 'PROOF=T',
network_owner=>'RDFUSER', network_name=>'NET1');
```

Specifying `PROOF=T` causes a view to be created containing proof for each inferred triple. The view name is the inferred graph name prefixed by `RDFUSER.NET1#SEMI_`. Two relevant columns in this view are LINK_ID and EXPLAIN (the proof). The following example displays the LINK_ID value and proof of each generated triple (with LINK_ID values shortened for simplicity):

```
SELECT link_id || ' generated by ' || explain as
          triple_and_its_proof FROM RDFUSER.NET1#SEMI_OWLST_IDX;

TRIPLE_AND_ITS_PROOF
--------------------------------------------------------------------
8_5_5_4 generated by 4_D_5_5 : SYMM_SAMH_SYMM
8_4_5_4 generated by 8_5_5_4 4_D_5_5 : SAM_SAMH
. . .
```

A proof consists of one or more triple (link) ID values and the name of the rule that is applied on those triples:

link-id1 [*link-id2* ... l*ink-idn*] : *rule-name*

Example 3-4 Displaying Proof Information

To get the full subject, predicate, and object URIs for proofs, you can query the RDF graph view and the inferred graph view. Example 3-4 displays the LINK_ID value and associated triple contents using the RDF graph view SEMM_OWLTST and the inferred graph view SEMI_OWLTST_IDX.

```
SELECT to_char(x.triple.rdf_m_id, 'FMXXXXXXXXXXXXXXXX') ||'_'||
       to_char(x.triple.rdf_s_id, 'FMXXXXXXXXXXXXXXXX') ||'_'||
       to_char(x.triple.rdf_p_id, 'FMXXXXXXXXXXXXXXXX') ||'_'||
       to_char(x.triple.rdf_c_id, 'FMXXXXXXXXXXXXXXXX'),
       x.triple.get_triple()
  FROM (
   SELECT sdo_rdf_triple_s(
           t.canon_end_node_id,
           t.model_id,
           t.start_node_id,
           t.p_value_id,
           t.end_node_id) triple
     FROM (select * from rdfuser.net1#semm_owltst union all
           select * from rdfuser.net1#semi_owltst_idx
          ) t
    WHERE t.link_id IN ('4_D_5_5','8_5_5_4')
  ) x;

   LINK_ID  X.TRIPLE.GET_TRIPLE()(SUBJECT, PROPERTY, OBJECT)
----------  ---------------------------------------------------------------
4_D_5_5 SDO_RDF_TRIPLE('<http://example.com/name/John>', '<http://www.w3.org/2002/07/
owl#sameAs>', '<http://example.com/name/JohnQ>')
8_5_5_4 SDO_RDF_TRIPLE('<http://example.com/name/JohnQ>', '<http://www.w3.org/2002/07/
owl#sameAs>', '<http://example.com/name/John>')
```

In Example 3-4, for the proof entry 8_5_5_4 generated by 4_D_5_5 : SYMM_SAMH_SYMM for the triple with LINK_ID = 8_5_5_4, it is inferred from the triple with 4_D_5_5 using the symmetricity of `owl:sameAs`.

If the inference status is INCOMPLETE and if the last inference was generated without proof information, you cannot invoke SEM_APIS.CREATE_INFERRED_GRAPH with `PROOF=T`. In this case, you must first drop the inferred graph and create it again specifying `PROOF=T`.

3.2.5 Validating OWL RDF Graphs and Inferred Graphs

An OWL ontology may contain errors, such as unsatisfiable classes, instances belonging to unsatisfiable classes, and two individuals asserted to be same and different at the same time. You can use the SEM_APIS.VALIDATE_RDF_GRAPH and SEM_APIS.VALIDATE_INFERRED_GRAPH functions to detect inconsistencies in the original RDF graph and in the inferred graph, respectively.

Example 3-5 Validating an Inferred Graph

Example 3-5 shows uses the SEM_APIS.VALIDATE_INFERRED_GRAPH function, which returns a null value if no errors are detected or a VARRAY of strings if any errors are detected.

```
SQL>  -- Insert an offending triple.
SQL>  insert into owltst values (1, sdo_rdf_triple_s('owltst',
             'urn:C1', 'http://www.w3.org/2000/01/rdf-schema#subClassOf', 'http://
```

```
www.w3.org/2002/07/owl#Nothing', 'RDFUSER', 'NET1'));

SQL> -- Drop inferred graph first.
SQL>  exec sem_apis.drop_inferred_graph('owltst_idx', network_owner=>'RDFUSER',
network_name=>'NET1');
PL/SQL procedure successfully completed.

SQL> -- Perform OWL inferencing.
SQL> exec sem_apis.create_inferred_graph('owltst_idx', sem_models('OWLTST'),
sem_rulebases('OWLPRIME') , network_owner=>'RDFUSER', network_name=>'NET1');
PL/SQL procedure successfully completed.

SQL > set serveroutput on;
SQL > -- Now invoke validation API: sem_apis.validate_inferred_graph
SQL >
declare
  lva sem_longvarchararray;
  idx int;
begin
  lva := sem_apis.validate_inferred_graph(sem_models('OWLTST'),
sem_rulebases('OWLPRIME'), network_owner=>'RDFUSER', network_name=>'NET1') ;

  if (lva is null) then
   dbms_output.put_line('No errors found.');
  else
    for idx in 1..lva.count loop
      dbms_output.put_line('Offending entry := ' || lva(idx)) ;
    end loop ;
  end if;
end ;
/

SQL> -- NOTE: The LINK_ID value and the numbers in the following
SQL> -- line are shortened for simplicity in this example. --

          Offending entry  := 1 10001 (4_2_4_8 2 4 8) Unsatisfiable class.
```

Each item in the validation report array includes the following information:

- Number of triples that cause this error (1 in Example 3-5)
- Error code (10001 Example 3-5)
- One or more triples (shown in parentheses in the output; (4_2_4_8 2 4 8) in Example 3-5).

  These numbers are the LINK_ID value and the ID values of the subject, predicate, and object.
- Descriptive error message (Unsatisfiable class. in Example 3-5)

The output in Example 3-5 indicates that the error is caused by one triple that asserts that a class is a subclass of an empty class owl:Nothing.

3.2.6 Using SEM_APIS.CREATE_INFERRED_GRAPH for RDFS Inference

In addition to accepting OWL vocabularies, the SEM_APIS.CREATE_INFERRED_GRAPH procedure accepts RDFS rulebases. The following example shows RDFS inference (all standard RDFS rules are defined in http://www.w3.org/TR/rdf-mt/):

```
EXECUTE sem_apis.create_inferred_graph('rdfstst_idx', sem_models('my_model'),
sem_rulebases('RDFS'), network_owner=>'RDFUSER', network_name=>'NET1');
```

Because rules RDFS4A, RDFS4B, RDFS6, RDFS8, RDFS10, RDFS13 may not generate meaningful inference for your applications, you can deselect those components for faster inference. The following example deselects these rules.

```
EXECUTE sem_apis.create_inferred_graph('rdfstst_idx', sem_models('my_model'),
sem_rulebases('RDFS'), SEM_APIS.REACH_CLOSURE, -
 'RDFS4A-, RDFS4B-, RDFS6-, RDFS8-, RDFS10-, RDFS13-'), network_owner=>'RDFUSER',
network_name=>'NET1');
```

3.2.7 Enhancing Inference Performance

This section describes suggestions for improving the performance of inference operations.

- Collect statistics before inferencing. After you load a large RDF/OWL data model, you should execute the SEM_PERF.GATHER_STATS procedure. See the Usage Notes for that procedure (in SEM_PERF Package Subprograms) for important usage information.
- Allocate sufficient temporary tablespace for inference operations. OWL inference support in Oracle relies heavily on table joins, and therefore uses significant temporary tablespace.
- Use the appropriate implementations of the SVFH and AVFH inference components.

  The default implementations of the SVFH and AVFH inference components work best when the number of restriction classes defined by `owl:someValuesFrom` and/or `owl:allValuesFrom` is low (as in the LUBM data sets). However, when the number of such classes is high (as in the Gene Ontology `http://www.geneontology.org/`), using non-procedural implementations of SVFH and AVFH may significantly improve performance.

  To disable the procedural implementations and to select the non-procedural implementations of SVFH and AVFH, include `'PROCSVFH=F'` and/or `'PROCAVFH=F'` in the options to SEM_APIS.CREATE_INFERRED_GRAPH. Using the appropriate implementation for an ontology can provide significant performance benefits. For example, selecting the non-procedural implementation of SVFH for the NCI Thesaurus ontology (see `http://www.cancer.gov/research/resources/terminology`) produced a 960% performance improvement for the SVFH inference component (tested on a dual-core, 8GB RAM desktop system with 3 SATA disks tied together with Oracle ASM).

See also Optimizing owl:sameAs Inference.

Related Topics

- Optimizing owl:sameAs Inference

3.2.8 Optimizing owl:sameAs Inference

You can optimize inference performance for large `owl:sameAs` cliques by specifying `'OPT_SAMEAS=T'` in the `options` parameter when performing OWLPrime inference. (A **clique** is a graph in which every node of it is connected to, bidirectionally, every other node in the same graph.)

According to OWL semantics, the `owl:sameAs` construct is treated as an equivalence relation, so it is reflexive, symmetric, and transitive. As a result, during inference a full materialization of `owl:sameAs`-related inferences could significantly increase the size of the inferred graph. Consider the following example triple set:

```
:John  owl:sameAs  :John1 .
:John  owl:sameAs  :John2 .
:John2 :hasAge     "32" .
```

Applying OWLPrime inference (with the `SAM` component specified) to this set would generate the following new triples:

```
:John1  owl:sameAs  :John .
:John2  owl:sameAs  :John .
:John1  owl:sameAs  :John2 .
:John2  owl:sameAs  :John1 .
:John   owl:sameAs  :John .
:John1  owl:sameAs  :John1 .
:John2  owl:sameAs  :John2 .
:John   :hasAge     "32" .
:John1  :hasAge     "32" .
```

In the preceding example, `:John`, `:John1` and `:John2` are connected to each other with the `owl:sameAs` relationship; that is, they are members of an `owl:sameAs` **clique**. To provide optimized inference for large `owl:sameAs` cliques, you can consolidate `owl:sameAs` triples without sacrificing correctness by specifying `'OPT_SAMEAS=T'` in the `options` parameter when performing OWLPrime inference. For example:

```
EXECUTE sem_apis.create_inferred_graph('M_IDX',sem_models('M'),
   sem_rulebases('OWLPRIME'),null,null,'OPT_SAMEAS=T', network_owner=>'RDFUSER',
network_name=>'NET1');
```

When you specify this option, for each `owl:sameAs` clique, one resource from the clique is chosen as a canonical representative and all of the inferences for that clique are consolidated around that resource. Using the preceding example, if `:John1` is the clique representative, after consolidation the inferred graph would contain only the following triples:

```
:John1 owl:sameAs :John1 .
:John1 :hasAge    "32" .
```

Some overhead is incurred with `owl:sameAs` consolidation. During inference, all asserted RDF graphs are copied into the inference partition, where they are consolidated together with the inferred triples. Additionally, for very large asserted graphs, consolidating and removing duplicate triples incurs a large runtime overhead, so the `OPT_SAMEAS=T` option is recommended only for ontologies that have a large number of `owl:sameAs` relationships and large clique sizes.

After the OPT_SAMEAS=T option has been used for an inferred graph, all subsequent uses of SEM_APIS.CREATE_INFERRED_GRAPH for that inferred graph must also use `OPT_SAMEAS=T`, or an error will be reported. To disable optimized `sameAs` handling, you must first drop the inferred graph.

Clique membership information is stored in a view named SEMC_*inferred-graph-name*, where *inferred-graph-name* is the name of the inferred graph. Each SEMC_*inferred-graph-name* view has the columns shown in Table 3-3.

Table 3-3   SEMC_inferred_graph_name View Columns

Column Name	Data Type	Description
MODEL_ID	NUMBER	ID number of the inferred model
VALUE_ID	NUMBER)	ID number of a resource that is a member of the `owl:sameAs` clique identified by CLIQUE_ID
CLIQUE_ID	NUMBER	ID number of the clique representative for the VALUE_ID resource

To save space, the SEMC_*inferred-graph-name* view does not contain reflexive rows like (CLIQUE_ID, CLIQUE_ID).

- Querying owl:sameAs Consolidated Inference Graphs

3.2.8.1 Querying owl:sameAs Consolidated Inference Graphs

At query time, if the inferred graph queried was created using the `OPT_SAMEAS=T` option, the results are returned from an `owl:sameAs`-consolidated inference partition. The query results are not expanded to include the full `owl:sameAs` closure.

In the following example query, the only result returned would be `:John1`, which is the canonical clique representative.

```
SELECT A FROM TABLE (
  SEM_MATCH ('SELECT ?A WHERE {?A :hasAge "32"}',SEM_MODELS('M'),
    SEM_RULEBASES('OWLPRIME'),null, null, null, null, 'PLUS_RDFT=VC', null, null,
'RDFUSER', 'NET1'));
```

With the preceding example, even though `:John2 :hasAge "32"` occurs in the RDF graph, it has been replaced during the inference consolidation phase where redundant triples are removed. However, you can expand the query results by performing a join with the RDFUSER.NET1#SEMC_*rules-index-name* view that contains the consolidated `owl:sameAs` information. For example, to get expanded result set for the preceding SEM_MATCH query, you can use the following expanded query:

```
SELECT V.VALUE_NAME A_VAL FROM TABLE (
  SEM_MATCH ('SELECT ?A WHERE {?A :hasAge "32"}',SEM_MODELS('M'),
    SEM_RULEBASES('OWLPRIME'), null, null, null, null, 'PLUS_RDFT=VC', null, null,
'RDFUSER', 'NET1')) Q,
    RDFUSER.NET1#RDF_VALUE$ V, RDFUSER.NET1#SEMC_M_IDX C
  WHERE V.VALUE_ID  = C.VALUE_ID
     AND C.CLIQUE_ID = Q.A$RDFVID
  UNION ALL
    SELECT A A_VAL FROM TABLE (
      SEM_MATCH ('SELECT ?A WHERE {?A :hasAge "32"}',SEM_MODELS('M'),
        SEM_RULEBASES('OWLPRIME'), null, null, null, null, 'PLUS_RDFT=VC', null, null,
'RDFUSER', 'NET1'));
```

Or, you could rewrite the preceding expanded query using a left outer join, as follows:

```
SELECT V.VALUE_NAME A_VAL FROM TABLE (
  SEM_MATCH ('(?A <http://hasAge> "33")',SEM_MODELS('M'),
   SEM_RULEBASES('OWLPRIME'), null, null, null, null, 'PLUS_RDFT=VC', null, null,
'RDFUSER', 'NET1')) Q,
  RDFUSER.NET1#RDF_VALUE$ V,
   (SELECT value_id, clique_id FROM RDFUSER.NET1#SEMC_M_IDX
  UNION ALL
    SELECT DISTINCT clique_id, clique_id
      FROM RDFUSER.NET1#SEMC_M_IDX) C
 WHERE Q.A$RDFVID  = c.clique_id  (+)
   AND V.VALUE_ID  = nvl(C.VALUE_ID, Q.A$RDFVID);
```

3.2.9 Performing Incremental Inference

Incremental inference can be used to update inferred graphs efficiently after triple additions. There are two ways to enable incremental inference for an inferred graph:

- Specify the `options` parameter value `INC=T` when creating the inferred graph. For example:

```
EXECUTE sem_apis.create_inferred_graph ('M_IDX',sem_models('M'),
  sem_rulebases('OWLPRIME'),null,null, 'INC=T', network_owner=>'RDFUSER',
network_name=>'NET1');
```

- Use the SEM_APIS.ENABLE_INC_INFERENCE procedure.

  If you use this procedure, the inferred graph must have a VALID status. Before calling the procedure, if you do not own the RDF graphs involved in the inferred graph, you must ensure that the respective RDF graph owners have used the SEM_APIS.ENABLE_CHANGE_TRACKING procedure to enable change tracking for those RDF graphs.

When incremental inference is enabled for an inferred graph, the parameter `INC=T` must be specified when invoking the SEM_APIS.CREATE_INFERRED_GRAPH procedure for that inferred graph.

Incremental inference for an inferred graph depends on triggers for the application tables of the RDF graphs involved in creating the inferred graph. This means that incremental inference works only when triples are inserted in the application tables underlying the inferred graph using conventional path loads, unless you specify the triples by using the `delta_in` parameter in the call to the SEM_APIS.CREATE_INFERRED_GRAPH procedure, as in the following example, in which the triples from RDF graph `M_NEW` will be added to the RDF graph `M`, and inferred graph `M_IDX` will be updated with the new inferences:

```
EXECUTE sem_apis.create_inferred_graph('M_IDX', sem_models('M'),
  sem_rulebases('OWLPRIME''), SEM_APIS.REACH_CLOSURE, null, null,
  sem_models('M_NEW'), network_owner=>'RDFUSER', network_name=>'NET1');
```

If multiple RDF graphs are involved in the incremental inference call, then to specify the destination RDF graph to which the `delta_in` RDF graph or RDF graphs are to be added, specify `DEST_MODEL=`*`<rdf_graph_name>`* in the `options` parameter. For example, the following causes the RDF data in RDF graph `M_NEW` to be added to the RDF graph `M2`:

```
EXECUTE sem_apis.create_inferred_graph('M_IDX', sem_models('M1','M2','M3'),
sem_rulebases('OWLPRIME''), SEM_APIS.REACH_CLOSURE, null, 'DEST_MODEL=M2',
sem_models('M_NEW')), network_owner=>'RDFUSER', network_name=>'NET1');
```

Another way to bypass the conventional path loading requirement when using incremental inference is to set the UNDO_RETENTION parameter to cover the intervals between inferred graphs when you perform bulk loading. For example, if the last inferred graph was created 6 hours ago, the UNDO_RETENTION value should be set to greater than 6 hours; if it is less than that, then (given a heavy workload and limited undo space) it is not guaranteed that all relevant undo information will be preserved for incremental inference to apply. In such cases, the SEM_APIS.CREATE_INFERRED_GRAPH procedure falls back to regular (non-incremental) inference.

To check if change tracking is enabled on an RDF graph, use the SEM_APIS.GET_CHANGE_TRACKING_INFO procedure. To get additional information about incremental inference for an inferred graph, use the SEM_APIS.GET_INC_INF_INFO procedure.

The following restrictions apply to incremental inference:

- It does not work with optimized `owl:sameAs` handling (`OPT_SAMEAS`), user-defined rules, VPD-enabled RDF graphs, or version-enabled RDF graphs.
- It supports only the addition of triples. With updates or deletions, the inferred graph will be completely rebuilt.
- It depends on triggers on application tables.

- Column types (RAW8 or NUMBER) used in incremental inference must be consistent. For instance, if `RAW8=T` is used to build the inferred graph initially, then for every subsequent SEM_APIS.CREATE_INFERRED_GRAPH call the same option must be used. To change the column type to NUMBER, you must drop and rebuild the inferred graph.

3.2.10 Using Parallel Inference

Parallel inference can improve inference performance by taking advantage of the capabilities of a multi-core or multi-CPU architectures. To use parallel inference, specify the `DOP` (degree of parallelism) keyword and an appropriate value when using the SEM_APIS.CREATE_INFERRED_GRAPH procedure. For example:

```
EXECUTE sem_apis.create_inferred_graph('M_IDX',sem_models('M'),
      sem_rulebases('OWLPRIME'), sem_apis.REACH_CLOSURE, null, 'DOP=4',
      network_owner=>'RDFUSER', network_name=>'NET1');
```

Specifying the DOP keyword causes parallel execution to be enabled for an Oracle-chosen set of inference components

The success of parallel inference depends heavily on a good hardware configuration of the system on which the database is running. The key is to have a "balanced" system that implements the best practices for database performance tuning and Oracle SQL parallel execution. For example, do not use a single 1 TB disk for an 800 GB database, because executing SQL statements in parallel on a single physical disk can even be slower than executing SQL statements in serial mode. Parallel inference requires ample memory; for each CPU core, you should have at least 4 GB of memory.

Parallel inference is best suited for large ontologies; however, inference performance can also improve for small ontologies.

There is some transient storage overhead associated with using parallel inference. Parallel inference builds a source table that includes all triples based on all the source RDF/OWL graphs and existing inferred graph. This table might use an additional 10 to 30 percent of storage compared to the space required for storing data and index of the source RDF graphs.

3.2.11 Using Named Graph Based Inferencing (Global and Local)

The default inferencing in Oracle Database takes all asserted triples from all the source RDF graph or RDF graphs provided and applies semantic rules on top of all the asserted triples until an inference closure is reached. Even if the given source RDF graphs contain one or more multiple named graphs, it makes no difference because all assertions, whether part of a named graph or not, are treated the same as if they come from a single graph. (For an introduction to named graph support in RDF Graph, see Named Graphs.)

This default inferencing can be thought of as completely "global" in that it does not consider named graphs at all.

However, if you use named graphs, you can override the default inferencing and have named graphs be considered by using either of the following features:

- Named graph based *global* inference (NGGI), which treats all specified named graphs as a unified graph. NGGI lets you narrow the scope of triples to be considered, while enabling great flexibility; it is explained in Named Graph Based Global Inference (NGGI).
- Named graph based *local* inference (NGLI), which treats each specified named graph as a separate entity. NGLI is explained in Named Graph Based Local Inference (NGLI).

For using NGGI and NGLI together, see a recommended usage flow in Using NGGI and NGLI Together.

You specify NGGI or NGLI through certain parameters and options to the SEM_APIS.CREATE_INFERRED_GRAPH procedure when you create an inferred graph.

- Named Graph Based Global Inference (NGGI)
- Named Graph Based Local Inference (NGLI)
- Using NGGI and NGLI Together

3.2.11.1 Named Graph Based Global Inference (NGGI)

Named graph based global inference (NGGI) enables you to narrow the scope of triples used for inferencing at the named graph level (as opposed to the RDF graph level). It also enables great flexibility in selecting the scope; for example, you can include triples from zero or more named graphs and/or from the default graph, and you can include all triples with a null graph name from specified RDF graphs.

For example, in a hospital application you may only want to apply the inference rules on all the information contained in a set of named graphs describing patients of a particular hospital. If the patient-related named graphs contains only instance-related assertions (ABox), you can specify one or multiple additional schema related-RDF graphs (TBox), as in Example 3-6.

Example 3-6 Named Graph Based Global Inference

```
EXECUTE sem_apis.create_inferred_graph(
  'patients_inf',
  rdf_graphs_in      => sem_models('patients','hospital_ontology'),
  rulebases_in       => sem_rulebases('owl2rl'),
  passes             => SEM_APIS.REACH_CLOSURE,
  inf_components_in  => null,
  options            => 'DOP=4,RAW8=T',
  include_default_g  => sem_models('hospital_ontology'),
  include_named_g    => sem_graphs('<urn:hospital1_patient1>','<urn:hospital1_patient2>'),
  inf_ng_name        => '<urn:inf_graph_for_hospital1>',
  network_owner      =>'RDFUSER',
  network_name       =>'NET1'
  );
```

In Example 3-6:

- Two RDF graphs are involved: `patients` contains a set of named graphs where each named graph holds triples relevant to a particular patient, and `hospital_ontology` contains schema information describing concepts and relationships that are defined for hospitals. These two RDF graphs together are the source graphs, and they set up an overall scope for the inference.
- The `include_default_g` parameter causes all triples with a NULL graph name in the specified RDF graphs to participate in NGGI. In this example, all triples with a NULL graph name in RDF graph `hospital_ontology` will be included in NGGI.
- The `include_named_g` parameter causes all triples from the specified named graphs (across all source RDF graphs) to participate in NGGI. In this example, triples from named graphs `<urn:hospital1_patient1>` and `<urn:hospital1_patient2>` will be included in NGGI.
- The `inf_ng_name` parameter assigns graph name `<urn:inf_graph_for_hospital1>` to all the new triples inferred by NGGI.

3.2.11.2 Named Graph Based Local Inference (NGLI)

Named graph based local inference (NGLI) treats each named graph as a separate entity instead of viewing the graphs as a single unified graph. Inference logic is performed within the boundary of each entity. You can specify schema-related assertions (TBox) in a default graph, and that default graph will participate the inference of each named graph. For example, inferred triples based on a graph with name `G1` will be assigned the same graph name `G1` in the inferred data partition.

Assertions from any two separate named graphs will never jointly produce any new assertions.

For example, assume the following:

- Graph `G1` includes the following assertion:

  ```
  :John  :hasBirthMother  :Mary .
  ```

- Graph `G2` includes the following assertion:

  ```
  :John  :hasBirthMother  :Bella .
  ```

- The default graph includes the assertion that `:hasBirthMother` is an `owl:FunctionalProperty`. (This assertion has a null graph name.)

In this example, named graph based *local* inference (NGLI) will ***not*** infer that `:Mary` is `owl:sameAs :Bella` because the two assertions are from two distinct graphs, `G1` and `G2`. By contrast, a named graph based *global* inference (NGGI) that includes `G1`, `G2`, and the functional property definition *would* be able to infer that `:Mary` is `owl:sameAs :Bella`.

NGLI currently does not work together with proof generation, user-defined rules, optimized `owl:sameAs` handling, or incremental inference.

Example 3-7 Named Graph Based Local Inference

Example 3-7 shows NGLI.

```
EXECUTE sem_apis.create_inferred_graph(
  'patients_inf',
  rdf_graphs_in     => sem_models('patients','hospital_ontology'),
  rulebases_in      => sem_rulebases('owl2rl'),
  passes            => SEM_APIS.REACH_CLOSURE,
  inf_components_in => null,
  options           => 'LOCAL_NG_INF=T',
  network_owner=>'RDFUSER',
  network_name=>'NET1'
);
```

In Example 3-7:

- The two RDF graphs patients and hospital_ontology together are the source graphs, and they set up an overall scope for the inference, similar to the case of global inference in Example 3-6. All triples with a null graph name are treated as part of the common schema (TBox). Inference is performed within the boundary of every single named graph combined with the common schema.
- Then `options` parameter keyword-value pair `LOCAL_NG_INF=T` specifies that named graph based local inference (NGLI) is to be performed.

Note that, by design, NGLI does not apply to the default graph itself. However, you can easily apply named graph based global inference (NGGI) on the default graph and set the `inf_ng_name` parameter to null. In this way, the TBox inference is precomputed, improving the overall performance and storage consumption.

NGLI does not allow the following:

- Inferring new relationships based on a mix of triples from multiple named graphs
- Inferring new relationships using only triples from the default graph.

To get the inference that you would normally expect, you should keep schema assertions and instance assertions separate. Schema assertions (for example, `:A rdfs:subClassOf :B` and `:p1 rdfs:subPropertyOf :p2`) should be stored in the default graph as unnamed triples (with null graph names). By contrast, instance assertions (for example, `:X :friendOf :Y`) should be stored in one of the named graphs.

For a discussion and example of using NGLI to perform document-centric inference with semantically indexed documents, see Performing Document-Centric Inference.

3.2.11.3 Using NGGI and NGLI Together

The following is a recommended usage flow for using NGGI and NGLI together. It assumes that TBox and ABox are stored in two separate RDF graphs, that TBox contains schema definitions and all triples in the TBox have a null graph name, but that ABox consists of a set of named graphs describing instance-related data.

1. Invoke NGGI on the TBox by itself. For example:

```
EXECUTE sem_apis.create_inferred_graph(
    'TEST_INF',
    sem_models('abox','tbox'),
    sem_rulebases('owl2rl'),
    SEM_APIS.REACH_CLOSURE,
    include_default_g=>sem_models('tbox'),
    network_owner=>'RDFUSER',
    network_name=>'NET1'
);
```

2. Invoke NGLI for all named graphs. For example:

```
EXECUTE sem_apis.create_inferred_graph(
    'TEST_INF',
    sem_models('abox','tbox'),
    sem_rulebases('owl2rl'),
    SEM_APIS.REACH_CLOSURE,
    options => 'LOCAL_NG_INF=T,ENTAIL_ANYWAY=T',
           network_owner=>'RDFUSER',
           network_name=>'NET1'
);
```

`ENTAIL_ANYWAY=T` is specified because the NGGI call in step 1 will set the status of inferred graph to `VALID`, and the SEM_APIS.CREATE_INFERRED_GRAPH procedure call in step 2 will quit immediately unless `ENTAIL_ANYWAY=T` is specified.

3.2.12 Performing Selective Inferencing (Advanced Information)

Selective inferencing is component-based inferencing, in which you limit the inferencing to specific OWL components that you are interested in. To perform selective inferencing, use the `inf_components_in` parameter to the SEM_APIS.CREATE_INFERRED_GRAPH procedure to specify a comma-delimited list of components. The final inferencing is determined by the *union* of rulebases specified and the components specified.

Example 3-8 Performing Selective Inferencing

Example 3-8 limits the inferencing to the class hierarchy from subclass (SCOH) relationship and the property hierarchy from subproperty (SPOH) relationship. This example creates an empty rulebase and then specifies the two components (`'SCOH,SPOH'`) in the call to the SEM_APIS.CREATE_INFERRED_GRAPH procedure.

```
EXECUTE sem_apis.create_rulebase('my_rulebase', network_owner=>'RDFUSER',
network_name=>'NET1');

EXECUTE sem_apis.create_inferred_graph('owltst_idx', sem_models('owltst'),
sem_rulebases('my_rulebase'), SEM_APIS.REACH_CLOSURE, 'SCOH,SPOH',
network_owner=>'RDFUSER', network_name=>'NET1');
```

The following component codes are available: `SCOH`, `COMPH`, `DISJH`, `SYMMH`, `INVH`, `SPIH`, `MBRH`, `SPOH`, `DOMH`, `RANH`, `EQCH`, `EQPH`, `FPH`, `IFPH`, `DOM`, `RAN`, `SCO`, `DISJ`, `COMP`, `INV`, `SPO`, `FP`, `IFP`, `SYMM`, `TRANS`, `DIF`, `SAM`, `CHAIN`, `HASKEY`, `ONEOF`, `INTERSECT`, `INTERSECTSCOH`, `MBRLST`, `PROPDISJH`, `SKOSAXIOMS`, `SNOMED`, `SVFH`, `THINGH`, `THINGSAM`, `UNION`, `RDFP1`, `RDFP2`, `RDFP3`, `RDFP4`, `RDFP6`, `RDFP7`, `RDFP8AX`, `RDFP8BX`, `RDFP9`, `RDFP10`, `RDFP11`, `RDFP12A`, `RDFP12B`, `RDFP12C`, `RDFP13A`, `RDFP13B`, `RDFP13C`, `RDFP14A`, `RDFP14BX`, `RDFP15`, `RDFP16`, `RDFS2`, `RDFS3`, `RDFS4a`, `RDFS4b`, `RDFS5`, `RDFS6`, `RDFS7`, `RDFS8`, `RDFS9`, `RDFS10`, `RDFS11`, `RDFS12`, `RDFS13`

The rules corresponding to components with a prefix of *RDFP* can be found in *Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary*, by H.J. Horst.

The syntax for deselecting a component is *component_name* followed by a minus (-) sign. For example, the following statement performs OWLPrime inference without calculating the `subClassOf` hierarchy:

```
EXECUTE sem_apis.create_inferred_graph('owltst_idx', sem_models('owltst'),
sem_rulebases('OWLPRIME'), SEM_APIS.REACH_CLOSURE, 'SCOH-', network_owner=>'RDFUSER',
network_name=>'NET1');
```

By default, the OWLPrime rulebase implements the transitive semantics of owl:sameAs. OWLPrime does not include the following rules (semantics):

```
U   owl:sameAs   V  .
U     p    X  .         ==>   V  p   X   .

U   owl:sameAs   V  .
X     p    U  .         ==>   X   p   V   .
```

The reason for not including these rules is that they tend to generate many assertions. If you need to include these assertions, you can include the `SAM` component code in the call to the SEM_APIS.CREATE_INFERRED_GRAPH procedure.

3.3 Using Semantic Operators to Query Relational Data

You can use semantic operators to query relational data in an ontology-assisted manner, based on the semantic relationship between the data in a table column and terms in an ontology.

The SEM_RELATED semantic operator retrieves rows based on semantic relatedness. The SEM_DISTANCE semantic operator returns distance measures for the semantic relatedness, so that rows returned by the SEM_RELATED operator can be ordered or restricted using the distance measure. The index type MDSYS.SEM_INDEXTYPE allows efficient execution of such queries, enabling scalable performance over large data sets.

> **Note:**
>
> SEM_RELATED and SEM_DISTANCE are not supported on schema-private RDF networks.

- Using the SEM_RELATED Operator
- Using the SEM_DISTANCE Ancillary Operator
- Creating a Semantic Index of Type MDSYS.SEM_INDEXTYPE
- Using SEM_RELATED and SEM_DISTANCE When the Indexed Column Is Not the First Parameter
- Using URIPREFIX When Values Are Not Stored as URIs

3.3.1 Using the SEM_RELATED Operator

Referring to the ontology example in Example: Disease Ontology, consider the following query that requires semantic matching: *Find all patients whose diagnosis is of the type 'Immune_System_Disorder'.* A typical database query of the PATIENTS table (described in Example: Disease Ontology) involving syntactic match will not return any rows, because no rows have a DIAGNOSIS column containing the exact value `Immune_System_Disorder`. For example the following query will not return any rows:

```
SELECT diagnosis FROM patients WHERE diagnosis = 'Immune_System_Disorder';
```

Example 3-9   SEM_RELATED Operator

However, many rows in the patient data table are relevant, because their diagnoses fall under this class. Example 3-9 uses the SEM_RELATED operator (instead of lexical equality) to retrieve all the relevant rows from the patient data table. (In this example, the term `Immune_System_Disorder` is prefixed with a namespace, and the default assumption is that the values in the table column also have a namespace prefix. However, that might not always be the case, as explained in Using URIPREFIX When Values Are Not Stored as URIs.)

```
SELECT diagnosis FROM patients
  WHERE SEM_RELATED (diagnosis,
    '<http://www.w3.org/2000/01/rdf-schema#subClassOf>',
    '<http://www.example.org/medical_terms/Immune_System_Disorder>',
    sem_models('medical_ontology'), sem_rulebases('owlprime')) = 1;
```

The SEM_RELATED operator has the following attributes:

```
SEM_RELATED(
  sub  VARCHAR2,
  predExpr  VARCHAR2,
  obj  VARCHAR2,
  ontologyName  SEM_MODELS,
  ruleBases  SEM_RULEBASES,
  index_status  VARCHAR2,
  lower_bound INTEGER,
  upper_bound INTEGER
 ) RETURN INTEGER;
```

The `sub` attribute is the name of table column that is being searched. The terms in the table column are typically the subject in a <subject, predicate, object> triple pattern.

The predExpr attribute represents the predicate that can appear as a label of the edge on the path from the subject node to the object node.

The obj attribute represents the term in the ontology for which related terms (related by the predExpr attribute) have to be found in the table (in the column specified by the sub attribute). This term is typically the object in a <subject, predicate, object> triple pattern. (In a query with the equality operator, this would be the query term.)

The ontologyName attribute is the name of the ontology that contains the relationships between terms.

The rulebases attribute identifies one or more rulebases whose rules have been applied to the ontology to infer new relationships. The query will be answered based both on relationships from the ontology and the inferred new relationships when this attribute is specified.

The index_status optional attribute lets you query the data even when the relevant inferred graph (created when the specified rulebase was applied to the ontology) does not have a valid status. If this attribute is null, the query returns an error if the inferred graph does not have a valid status. If this attribute is not null, it must be the string VALID, INCOMPLETE, or INVALID, to specify the minimum status of the inferred graph for the query to succeed. Because OWL does not guarantee monotonicity, the value INCOMPLETE should not be used when an OWL Rulebase is specified.

The lower_bound and upper_bound optional attributes let you specify a bound on the distance measure of the relationship between terms that are related. See Using the SEM_DISTANCE Ancillary Operator for the description of the distance measure.

The SEM_RELATED operator returns 1 if the two input terms are related with respect to the specified predExpr relationship within the ontology, and it returns 0 if the two input terms are not related. If the lower and upper bounds are specified, it returns 1 if the two input terms are related with a distance measure that is greater than or equal to lower_bound and less than or equal to upper_bound.

3.3.2 Using the SEM_DISTANCE Ancillary Operator

The SEM_DISTANCE ancillary operator computes the distance measure for the rows filtered using the SEM_RELATED operator. The SEM_DISTANCE operator has the following format:

```
SEM_DISTANCE (number) RETURN NUMBER;
```

The number attribute can be any number, as long as it matches the number that is the last attribute specified in the call to the SEM_RELATED operator (see Example 3-10). The number is used to match the invocation of the ancillary operator SEM_DISTANCE with a specific SEM_RELATED (primary operator) invocation, because a query can have multiple invocations of primary and ancillary operators.

Example 3-10   SEM_DISTANCE Ancillary Operator

Example 3-10 expands Example 3-9 to show several statements that include the SEM_DISTANCE ancillary operator, which gives a measure of how closely the two terms (here, a patient's diagnosis and the term Immune_System_Disorder) are related by measuring the distance between the terms. Using the ontology described in Example: Disease Ontology, the distance between AIDS and Immune_System_Disorder is 3.

```
SELECT diagnosis, SEM_DISTANCE(123) FROM patients
  WHERE SEM_RELATED (diagnosis,
    '<http://www.w3.org/2000/01/rdf-schema#subClassOf>',
    '<http://www.example.org/medical_terms/Immune_System_Disorder>',
    sem_models('medical_ontology'), sem_rulebases('owlprime'), 123) = 1;
```

```
SELECT diagnosis FROM patients
  WHERE SEM_RELATED (diagnosis,
    '<http://www.w3.org/2000/01/rdf-schema#subClassOf>',
    '<http://www.example.org/medical_terms/Immune_System_Disorder>',
    sem_models('medical_ontology'), sem_rulebases('owlprime'), 123) = 1
  ORDER BY SEM_DISTANCE(123);

SELECT diagnosis, SEM_DISTANCE(123) FROM patients
  WHERE SEM_RELATED (diagnosis,
    '<http://www.w3.org/2000/01/rdf-schema#subClassOf>',
    '<http://www.example.org/medical_terms/Immune_System_Disorder>',
    sem_models('medical_ontology'), sem_rulebases('owlprime'), 123) = 1
  AND SEM_DISTANCE(123) <= 3;
```

Example 3-11 Using SEM_DISTANCE to Restrict the Number of Rows Returned

Example 3-11 uses distance information to restrict the number of rows returned by the primary operator. All rows with a term related to the object attribute specified in the SEM_RELATED invocation, but with a distance of greater than or equal to 2 and less than or equal to 4, are retrieved.

```
SELECT diagnosis FROM patients
  WHERE SEM_RELATED (diagnosis,
    '<http://www.w3.org/2000/01/rdf-schema#subClassOf>',
    '<http://www.example.org/medical_terms/Immune_System_Disorder>',
    sem_models('medical_ontology'), sem_rulebases('owlprime'), 2, 4) = 1;
```

In Example 3-11, the lower and upper bounds are specified using the `lower_bound` and `upper_bound` parameters in the SEM_RELATED operator instead of using the SEM_DISTANCE operator. The SEM_DISTANCE operator can be also be used for restricting the rows returned, as shown in the last SELECT statement in Example 3-10.

- Computation of Distance Information

3.3.2.1 Computation of Distance Information

Distances are generated for the following properties during inference (inferred graph): OWL properties defined as transitive properties, and RDFS `subClassOf` and RDFS `subPropertyOf` properties. The distance between two terms linked through these properties is computed as the shortest distance between them in a hierarchical class structure. Distances of two terms linked through other properties are undefined and therefore set to null.

Each transitive property link in the original model (viewed as a hierarchical class structure) has a distance of 1, and the distance of an inferred triple is generated according to the number of links between the two terms. Consider the following hypothetical sample scenarios:

- If the original graph contains `C1 rdfs:subClassOf C2` and `C2 rdfs:subClassOf C3`, then `C1 rdfs:subClassof of C3` will be derived. In this case:
  - `C1 rdfs:subClassOf C2`: distance = 1, because it exists in the model.
  - `C2 rdfs:subClassOf C3`: distance = 1, because it exists in the model.
  - `C1 rdfs:subClassOf C3`: distance = 2, because it is generated during inference.
- If the original graph contains `P1 rdfs:subPropertyOf P2` and `P2 rdfs:subPropertyOf P3`, then P1 rdfs:subPropertyOf P3 will be derived. In this case:
  - `P1 rdfs:subPropertyOf P2`: distance = 1, because it exists in the model.
  - `P2 rdfs:subPropertyOf P3`: distance = 1, because it exists in the model.

  - `P1 rdfs:subPropertyOf P3`: distance = 2, because it is generated during inference.
- If the original graph contains `C1 owl:equivalentClass C2` and `C2 owl:equivalentClass C3`, then `C1 owl:equivalentClass C3` will be derived. In this case:
  - `C1 owl:equivalentClass C2`: distance = 1, because it exists in the model.
  - `C2 owl:equivalentClass C3`: distance = 1, because it exists in the model.
  - `C1 owl:equivalentClass C3`: distance = 2, because it is generated during inference.

The SEM_RELATED operator works with user-defined rulebases. However, using the SEM_DISTANCE operator with a user-defined rulebase is not yet supported, and will raise an error.

3.3.3 Creating a Semantic Index of Type MDSYS.SEM_INDEXTYPE

When using the SEM_RELATED operator, you can create a semantic index of type MDSYS.SEM_INDEXTYPE on the column that contains the ontology terms. Creating such an index will result in more efficient execution of the queries. The CREATE INDEX statement must contain the `INDEXTYPE IS MDSYS.SEM_INDEXTYPE` clause, to specify the type of index being created.

Example 3-12 Creating a Semantic Index

Example 3-12 creates a semantic index named DIAGNOSIS_SEM_IDX on the DIAGNOSIS column of the PATIENTS table using the ontology in Example: Disease Ontology.

```
CREATE INDEX diagnosis_sem_idx
  ON patients (diagnosis)
  INDEXTYPE IS MDSYS.SEM_INDEXTYPE;
```

The column on which the index is built (DIAGNOSIS in Example 3-12) must be the first parameter to the SEM_RELATED operator, in order for the index to be used. If it not the first parameter, the index is not used during the execution of the query.

Example 3-13 Creating a Semantic Index Specifying an RDF and Rulebase

To improve the performance of certain RDF queries, you can cause statistical information to be generated for the semantic index by specifying one or more RDF graphs and rulebases when you create the index. Example 3-13 creates an index that will also generate statistics information for the specified RDF graph and rulebase. The index can be used with other RDF graphs and rulebases during query, but the statistical information will be used only if the RDF graph and rulebase specified during the creation of the index are the same RDF graph and rulebase specified in the query.

```
CREATE INDEX diagnosis_sem_idx
  ON patients (diagnosis)
  INDEXTYPE IS MDSYS.SEM_INDEXTYPE('ONTOLOGY_MODEL(medical_ontology),
    RULEBASE(OWLPrime)');
```

Example 3-14 Query Benefitting from Generation of Statistical Information

The statistical information is useful for queries that return top-k results sorted by semantic distance. Example 3-14 shows such a query.

```
SELECT /*+ FIRST_ROWS */ diagnosis FROM patients
  WHERE SEM_RELATED (diagnosis,
    '<http://www.w3.org/2000/01/rdf-schema#subClassOf>',
    '<http://www.example.org/medical_terms/Immune_System_Disorder>',
    sem_models('medical_ontology'), sem_rulebases('owlprime'), 123) = 1
  ORDER BY SEM_DISTANCE(123);
```

3.3.4 Using SEM_RELATED and SEM_DISTANCE When the Indexed Column Is Not the First Parameter

If an index of type MDSYS.SEM_INDEXTYPE has been created on a table column that is the first parameter to the SEM_RELATED operator, the index will be used. For example, the following query retrieves all rows that have a value in the DIAGNOSIS column that is a subclass of (`rdfs:subClassOf`) `Immune_System_Disorder`.

```
SELECT diagnosis FROM patients
  WHERE SEM_RELATED (diagnosis,
    '<http://www.w3.org/2000/01/rdf-schema#subClassOf>',
    '<http://www.example.org/medical_terms/Immune_System_Disorder>',
    sem_models('medical_ontology'), sem_rulebases('owlprime')) = 1;
```

Assume, however, that this query instead needs to retrieve all rows that have a value in the DIAGNOSIS column for which `Immune_System_Disorder` is a subclass. You could rewrite the query as follows:

```
SELECT diagnosis FROM patients
  WHERE SEM_RELATED
    ('<http://www.example.org/medical_terms/Immune_System_Disorder>',
    '<http://www.w3.org/2000/01/rdf-schema#subClassOf>',
    diagnosis,
    sem_models('medical_ontology'), sem_rulebases('owlprime')) = 1;
```

However, in this case a semantic index on the DIAGNOSIS column will not be used, because it is not the first parameter to the SEM_RELATED operator. To cause the index to be used, you can change the preceding query to use the `inverseOf` keyword, as follows:

```
SELECT diagnosis FROM patients
  WHERE SEM_RELATED (diagnosis,
    'inverseOf(http://www.w3.org/2000/01/rdf-schema#subClassOf)',
    '<http://www.example.org/medical_terms/Immune_System_Disorder>',
    sem_models('medical_ontology'), sem_rulebases('owlprime')) = 1;
```

This form causes the table column (on which the index is built) to be the first parameter to the SEM_RELATED operator, and it retrieves all rows that have a value in the DIAGNOSIS column for which `Immune_System_Disorder` is a subclass.

3.3.5 Using URIPREFIX When Values Are Not Stored as URIs

By default, the semantic operator support assumes that the values stored in the table are URIs. These URIs can be from different namespaces. However, if the values in the table do not have URIs, you can use the URIPREFIX keyword to specify a URI when you create the semantic index. In this case, the specified URI is prefixed to the value in the table and stored in the index structure. (One implication is that multiple URIs cannot be used).

Example 3-15 creates a semantic index that uses a URI prefix.

Example 3-15 Specifying a URI Prefix During Semantic Index Creation

```
CREATE INDEX diagnosis_sem_idx
  ON patients (diagnosis)
  INDEXTYPE IS MDSYS.SEM_INDEXTYPE
  PARAMETERS('URIPREFIX(<http://www.example.org/medical/>)');
```

The slash (/) character at the end of the URI is important, because the URI is prefixed to the table value (in the index structure) without any parsing.

4

Simple Knowledge Organization System (SKOS) Support

You can perform inferencing based on a core subset of the Simple Knowledge Organization System (SKOS) data model, which is especially useful for representing thesauri, classification schemes, taxonomies, and other types of controlled vocabulary.

SKOS is based on standard semantic web technologies including RDF and OWL, which makes it easy to define the formal semantics for those knowledge organization systems and to share the semantics across applications.

Support is provided for most, but not all, of the features of SKOS, the detailed specification of which is available at `http://www.w3.org/TR/skos-reference/`.

Around 40 SKOS-specific terms are included in the RDF Graph support, such as `skos:broader`, `skos:relatedMatch`, and `skos:Concept`. Over 100 SKOS axiomatic triples have been added, providing the basic coverage of SKOS semantics. However, support is not included for the integrity conditions described in the SKOS specification.

To perform SKOS-based inferencing, specify the system-defined `SKOSCORE` rulebase in the `rulebases_in` parameter in the call to the SEM_APIS.CREATE_INFERRED_GRAPH procedure, as in the following example:

```
EXECUTE sem_apis.create_inferred_graph('tstidx',sem_models('tst'),
sem_rulebases('skoscore'), network_owner=>'RDFUSER', network_name=>'NET1');
```

Example 4-1 defines, in Turtle format, a simple electronics scheme and two relevant concepts, cameras and digital cameras. Its meaning is straightforward and its representation is in RDF. It can be managed by Oracle Database in the same way as other RDF and OWL data.

Example 4-1 SKOS Definition of an Electronics Scheme

```
ex1:electronicsScheme rdf:type skos:ConceptScheme;

ex1:cameras rdf:type skos:Concept;
   skos:prefLabel "cameras"@en;
   skos:inScheme ex1:electronicsScheme.

ex1:digitalCameras rdf:type skos:Concept;
   skos:prefLabel "digital cameras"@en;
   skos:inScheme ex1:electronicsScheme.

ex1:digitalCameras skos:broader ex1:cameras.
```

- Supported and Unsupported SKOS Semantics
  This section describes features of SKOS semantics that are and are not supported by Oracle Database.
- Performing Inference on SKOS RDF Graphs
  Performing inference on a SKOS RDF graph is similar to performing inference on an RDF graph.

4.1 Supported and Unsupported SKOS Semantics

This section describes features of SKOS semantics that are and are not supported by Oracle Database.

- Supported SKOS Semantics
- Unsupported SKOS Semantics

4.1.1 Supported SKOS Semantics

All terms defined in SKOS and SKOS extension for labels are recognized. When the SKOSCORE rulebase is chosen for inference, the recognized terms include the following:

```
skos:altLabel
skos:broader
skos:broaderTransitive
skos:broadMatch
skos:changeNote
skos:closeMatch
skos:Collection
skos:Concept
skos:ConceptScheme
skos:definition
skos:editorialNote
skos:exactMatch
skos:example
skos:hasTopConcept
skos:hiddenLabel
skos:historyNote
skos:inScheme
skos:mappingRelation
skos:member
skos:memberList
skos:narrower
skos:narrowerTransitive
skos:narrowMatch
skos:notation
skos:note
skos:OrderedCollection
skos:prefLabel
skos:related
skos:relatedMatch
skos:scopeNote
skos:semanticRelation
skos:topConceptOf
skosxl:altLabel
skosxl:hiddenLabel
skosxl:Label
skosxl:labelRelation
skosxl:literalForm
skosxl:prefLabel
```

Most SKOS axioms and definitions are supported including the following: S1-S8, S10-S11, S15-S26, S28-S31, S33-S36, S38-S45, S47-S50, and S53-S54. (See the SKOS detailed specification for definitions.)

Most SKOS integrity conditions are supported, including S9, S13, S27, S37, and S46.

S52 is partially supported.

S55, S56, and S57 are not supported by default.

- S55, the property chain (`skosxl:prefLabel, skosxl:literalForm`), is a subproperty of `skos:prefLabel`.
- S56, the property chain (`skosxl:altLabel, skosxl:literalForm`), is a subproperty of `skos:altLabel`.
- S57, the property chain (`skosxl:hiddenLabel, skosxl:literalForm`), is a subproperty of `skos:hiddenLabel.chains`.

However, S55, S56, and S57 can be implemented using the OWL 2 subproperty chain construct. For information about property chain handling, see Property Chain Handling.

4.1.2 Unsupported SKOS Semantics

The following features of SKOS semantics are not supported:

- S12 and S51: The `rdfs:range` of the relevant predicates is the class of RDF plain literals. There is no check that the object values of these predicates are indeed plain literals; however, applications can perform such a check.
- S14: A resource has no more than one value of `skos:prefLabel` per language tag. This integrity condition is even beyond OWL FULL semantics, and it is not enforced in the current release.
- S32: The `rdfs:range of skos:member` is the union of classes `skos:Concept` and `skos:Collection`. This integrity condition is not enforced.
- S55, S56, and S57 are not supported by default, but they can be implemented using the OWL 2 subproperty chain construct, as explained in Supported SKOS Semantics.

4.2 Performing Inference on SKOS RDF Graphs

Performing inference on a SKOS RDF graph is similar to performing inference on an RDF graph.

To create an SKOS RDF graph, use the same procedure (SEM_APIS.CREATE_RDF_GRAPH) as for creating an RDF graph. You can load data into an SKOS RDF graph in the same way as for RDF graphs.

To infer new relationships for one or more SKOS RDF graphs, use the SEM_APIS.CREATE_INFERRED_GRAPH procedure with the system-defined rulebase `SKOSCORE`. For example:

```
EXECUTE sem_apis.create_inferred_graph('tstidx',sem_models('tst'),
sem_rulebases('skoscore')), network_owner=>'RDFUSER', network_name=>'NET1');
```

The inferred data will include many of the axioms defined in the SKOS detailed specification. Like other system-defined rulebases, SKOSCORE has no explicit rules; all the semantics supported are coded into the implementation.

- Validating SKOS RDF Graphs and Inferred Graphs
- Property Chain Handling

4.2.1 Validating SKOS RDF Graphs and Inferred Graphs

You can use the SEM_APIS.VALIDATE_INFERRED_GRAPH and SEM_APIS.VALIDATE_RDF_GRAPH procedures to validate the supported integrity conditions.

The output will include any inconsistencies caused by the supported integrity conditions, such as OWL 2 `propertyDisjointWith` and `S52`.

Example 4-2 validates an SKOS inferred graph.

Example 4-2 Validating an SKOS Inferred Graph

```
set serveroutput on
declare
  lva sem_longvarchararray;
  idx int;
begin
  lva := sem_apis.validate_inferred_graph(sem_models('tstskos'),
sem_rulebases('skoscore'), network_owner=>'RDFUSER',network_name=>'NET1');
  if (lva is null) then
    dbms_output.put_line('No conflicts');
  else
  for idx in 1..lva.count loop
    dbms_output.put_line('entry ' || idx || ' ' || lva(idx));
  end loop;
  end if;
end;
 /
```

4.2.2 Property Chain Handling

The SKOS S55, S56, and S57 semantics are not supported by default. However, you can add support for them by using the OWL 2 subproperty chain construct.

Example 4-3 inserts the necessary chain definition triples for S55 into an SKOS model. After the insertion, an invocation of SEM_APIS.CREATE_INFERRED_GRAPH that specifies the `SKOSCORE` rulebase will include the semantics defined in S55.

Example 4-3 Property Chain Insertions to Implement S55

```
INSERT INTO tst VALUES(sdo_rdf_triple_s('tst','<http://www.w3.org/2004/02/skos/
core#prefLabel>', '<http://www.w3.org/2002/07/owl#propertyChainAxiom>', '_:jA1',
'RDFUSER', 'NET1'));
INSERT INTO tst VALUES(sdo_rdf_triple_s('tst','_:jA1', '<http://www.w3.org/1999/02/22-
rdf-syntax-ns#first>', '<http://www.w3.org/2008/05/skos-xl#prefLabel>', 'RDFUSER',
'NET1'));
INSERT INTO tst VALUES(sdo_rdf_triple_s('tst','_:jA1', '<http://www.w3.org/1999/02/22-
rdf-syntax-ns#rest>', '_:jA2', 'RDFUSER', 'NET1'));
INSERT INTO tst VALUES(sdo_rdf_triple_s('tst','_:jA2', '<http://www.w3.org/1999/02/22-
rdf-syntax-ns#first>', '<http://www.w3.org/2008/05/skos-xl#literalForm>', 'RDFUSER',
'NET1'));
INSERT INTO tst VALUES(sdo_rdf_triple_s('tst','_:jA2', '<http://www.w3.org/1999/02/22-
rdf-syntax-ns#rest>', '<http://www.w3.org/1999/02/22-rdf-syntax-ns#nil>', 'RDFUSER',
'NET1'));
```

5 Semantic Indexing for Documents

Information extractors locate and extract meaningful information from unstructured documents. The ability to search for documents based on this extracted information is a significant improvement over the keyword-based searches supported by the full-text search engines.

Semantic indexing for documents introduces an index type that can make use of information extractors and annotators to semantically index documents stored in relational tables. Documents indexed semantically can be searched using SEM_CONTAINS operator within a standard SQL query. The search criteria for these documents are expressed using SPARQL query patterns that operate on the information extracted from the documents, as in the following example.

```
SELECT docId
FROM   Newsfeed
WHERE  SEM_CONTAINS (article,
     ' { ?org    rdf:type            typ:Organization  .
         ?org    pred:hasCategory    cat:BusinessFinance } ', ..) = 1
```

The key components that facilitate Semantic Indexing for documents in an Oracle Database include:

- Extensible information extractor framework, which allows third-party information extractors to be plugged into the database
- SEM_CONTAINS operator to identify documents of interest, based on their extracted information, using standard SQL queries
- SEM_CONTAINS_SELECT ancillary operator to return relevant information about the documents identified using SEM_CONTAINS operator
- SemContext index type to interact with the information extractor and manage the information extracted from a document set in an index structure and to facilitate semantically meaningful searches on the documents

The application program interface (API) for managing extractor policies and semantic indexes created for documents is provided in the SEM_RDFCTX PL/SQL package. SEM_RDFCTX Package Subprograms provides the reference information about the subprograms in SEM_RDFCTX package.

- Information Extractors for Semantically Indexing Documents
  Information extractors process unstructured documents and extract meaningful information from them, often using natural-language processing engines with the aid of ontologies.
- Extractor Policies
  An **extractor policy** is a named dictionary entity that determines the characteristics of a semantic index that is created using the policy.
- Semantically Indexing Documents
  Textual documents stored in a CLOB or VARCHAR2 column of a relational table can be indexed using the MDSYS.SEMCONTEXT index type, to facilitate semantically meaningful searches.

- SEM_CONTAINS and Ancillary Operators
  You can use the SEM_CONTAINS operator in a standard SQL statement to search for documents or document references that are stored in relational tables.
- Searching for Documents Using SPARQL Query Patterns
  Documents that are semantically indexed (that is, indexed using the mdsys.SemContext index type) can be searched using SEM_CONTAINS operator within a standard SQL query.
- Bindings for SPARQL Variables in Matching Subgraphs in a Document (SEM_CONTAINS_SELECT Ancillary Operator)
  You can use the SEM_CONTAINS_SELECT ancillary operator to return additional information about each document matched using the SEM_CONTAINS operator.
- Improving the Quality of Document Search Operations
  The quality of a document search operation depends on the quality of the information produced by the extractor used to index the documents. If the information extracted is incomplete, you may want to add some annotations to a document.
- Indexing External Documents
  You can use semantic indexing on documents that are stored in a file system or on the network. In such cases, you store the references to external documents in a table column, and you create a semantic index on the column using an appropriate extractor policy.
- Configuring the Calais Extractor type
  The CALAIS_EXTRACTOR type, which is a subtype of the RDFCTX_WS_EXTRACTOR type, enables you to access a Web service end point anywhere on the network, including the one that is publicly accessible (`OpenCalais.com`).
- Working with General Architecture for Text Engineering (GATE)
  General Architecture for Text Engineering (GATE) is an open source natural language processor and information extractor.
- Creating a New Extractor Type
  You can create a new extractor type by extending the RDFCTX_EXTRACTOR or RDFCTX_WS_EXTRACTOR extractor type.
- Creating a Local Semantic Index on a Range-Partitioned Table
  A local index can be created on a VARCHAR2 or CLOB column of a range-partitioned table.
- Altering a Semantic Index
  You can use the ALTER INDEX statement with a semantic index.
- Passing Extractor-Specific Parameters in CREATE INDEX and ALTER INDEX
  The CREATE INDEX and ALTER INDEX statements allow the passing of parameters needed by extractors.
- Performing Document-Centric Inference
  Document-centric inference refers to the ability to infer from each document individually.
- Metadata Views for Semantic Indexing
  This section describes views that contain metadata about semantic indexing
- Default Style Sheet for GATE Extractor Output
  This section lists the default XML style sheet that the `mdsys.gatenlp_extractor` implementation uses to convert the annotation set (encoded in XML) into RDF/XML.

5.1 Information Extractors for Semantically Indexing Documents

Information extractors process unstructured documents and extract meaningful information from them, often using natural-language processing engines with the aid of ontologies.

The quality and the completeness of information extracted from a document vary from one extractor to another. Some extractors simply identify the entities (such as names of persons, organizations, and geographic locations from a document), while the others attempt to identify the relationships among the identified entities and additional description for those entities. You can search for a specific document from a large set when the information extracted from the documents is maintained as a semantic index.

You can use an information extractor to create a semantic index on the documents stored in a column of a relational table. An extensible framework allows any third-party information extractor that is accessible from the database to be plugged into the database. An object type created for an extractor encapsulates the extraction logic, and has methods to configure the extractor and receive information extracted from a given document in RDF/XML format.

An abstract type MDSYS.RDFCTX_EXTRACTOR defines the common interfaces to all information extractors. An implementation of this abstract type interacts with a specific information extractor to produce RDF/XML for a given document. An implementation for this type can access a third-party information extractor that either is available as a database application or is installed on the network (accessed using Web service callouts). Example 5-1 shows the definition of the RDFCTX_EXTRACTOR abstract type.

Example 5-1 RDFCTX_EXTRACTOR Abstract Type Definition

```
create or replace type rdfctx_extractor authid current_user as object (
  extr_type         VARCHAR2(32),
  member function   getDescription return VARCHAR2,
  member function   rdfReturnType return VARCHAR2,
  member function   getContext(attribute VARCHAR2) return VARCHAR2,
  member procedure startDriver,
  member function   extractRDF(document CLOB,
                               docId    VARCHAR2) return CLOB,
  member function   extractRdf(document CLOB,
                               docId    VARCHAR2,
                               params   VARCHAR2,
                               options  VARCHAR2 default NULL) return CLOB
  member function   batchExtractRdf(docCursor         SYS_REFCURSOR,
                               extracted_info_table  VARCHAR2,
                               params                VARCHAR2,
                               partition_name        VARCHAR2 default NULL,
                               docId                 VARCHAR2 default NULL,
                               preferences           SYS.XMLType default NULL,
                               options               VARCHAR2 default NULL)
                               return CLOB,
  member procedure closeDriver
) not instantiable not final
/
```

A specific implementation of the RDFCTX_EXTRACTOR type sets an identifier for the extractor type in the `extr_type` attribute, and it returns a short description for the extractor type using `getDescription` method. All implementations of this abstract type return the extracted information as RDF triples. In the current release, the RDF triples are expected to be serialized using RDF/XML format, and therefore the `rdfReturnType` method should return '`RDF/XML`'.

An extractor type implementation uses the `extractRDF` method to encapsulate the extraction logic, possibly by invoking external information extractor using proprietary interfaces, and returns the extracted information in RDF/XML format. When a third-party extractor uses some proprietary XML Schema to capture the extracted information, an XML style sheet can be used to generate an equivalent RDF/XML. The `startDriver` and `closeDriver` methods can perform any housekeeping operations pertaining to the information extractor. The optional `params` parameter allows the extractor to obtain additional information about the type of extraction needed (for example, the desired quality of extraction).

Optionally, an extractor type implementation may support a batch interface by providing an implementation of the `batchExtractRdf` member function. This function accepts a cursor through the input parameter `docCursor` and typically uses that cursor to retrieve each document, extract information from the document, and then insert the extracted information into (the specified partition identified by the `partition_name` partition of the `extracted_info_table` table. The `preferences` parameter is used to obtain the preferences value associated with the policy (as described in Indexing External Documents and in the SEM_RDFCTX.CREATE_POLICY reference section).

The `getContext` member function accepts an attribute name and returns the value for that attribute. Currently this function is used only for extractors supporting the batch interface. The attribute names and corresponding possible return values are the following:

- For the `BATCH_SUPPORT` attribute, the return values are '`YES`' or '`NO`' depending on whether the extractor supports the batch interface.
- For the `DBUSER` attribute, the return value is the name of a database user that will connect to the database to retrieve rows from the cursor (identified by the `docCursor` parameter) and that will write to the table `extracted_info_table`.

This information is used for granting appropriate privileges to the table being indexed and the table `extracted_info_table`.

The `startDriver` and `closeDriver` methods can perform any housekeeping operations pertaining to the information extractor.

An extractor type for the General Architecture for Text Engineering (GATE) engine is defined as a subtype of the RDFCTX_EXTRACTOR type. The implementation of this extractor type sends the documents to a GATE engine over a TCP connection, receives annotations extracted by the engine in XML format, and converts this proprietary XML document to an RDF/XML document. For more information on configuring a GATE engine to work with Oracle Database, see Working with General Architecture for Text Engineering (GATE). For an example of creating a new information extractor, see Creating a New Extractor Type.

Information extractors that are deployed as Web services can be invoked from the database by extending the RDFCTX_WS_EXTRACTOR type, which is a subtype of the RDFCTX_EXTRACTOR type. The RDFCTX_WS_EXTRACTOR type encapsulates the Web service callouts in the `extractRDF` method; specific implementations for network-based extractors can reuse this implementation by setting relevant attribute values in the type constructor.

Thomson Reuters Calais is an example of a network-based information extractor that can be accessed using web-service callouts. The CALAIS_EXTRACTOR type, which is a subtype of the RDFCTX_WS_EXTRACTOR type, encapsulates the Calais extraction logic, and it can be used to semantically index the documents. The CALAIS_EXTRACTOR type must be configured for the database instance before it can be used to create semantic indexes, as explained in Configuring the Calais Extractor type.

5.2 Extractor Policies

An **extractor policy** is a named dictionary entity that determines the characteristics of a semantic index that is created using the policy.

Each extractor policy refers, directly or indirectly, to an instance of an extractor type. An extractor policy with a direct reference to an extractor type instance can be used to compose other extractor policies that include additional RDF graphs for ontologies.

The following example creates a basic extractor policy created using the GATE extractor type:

```
begin
  sem_rdfctx.create_policy (policy_name => 'SEM_EXTR',
                            extractor   => mdsys.gatenlp_extractor());
end;
/
```

The following example creates a dependent extractor policy that combines the metadata extracted by the policy in the preceding example with a user-defined RDF graph named `geo_ontology`:

```
begin
  sem_rdfctx.create_policy (policy_name => 'SEM_EXTR_PLUS_GEOONT',
                            base_policy => 'SEM_EXTR',
                            user_models => SEM_MODELS ('geo_ontology'));
end;
/
```

You can use an extractor policy to create one or more semantic indexes on columns that store unstructured documents, as explained in Semantically Indexing Documents.

5.3 Semantically Indexing Documents

Textual documents stored in a CLOB or VARCHAR2 column of a relational table can be indexed using the MDSYS.SEMCONTEXT index type, to facilitate semantically meaningful searches.

The extractor policy specified at index creation determines the information extractor used to semantically index the documents. The extracted information, captured as a set of RDF triples for each document, is managed in the RDF data store. Each instance of the semantic index is associated with a system-generated RDF graph, which maintains the RDF triples extracted from the corresponding documents.

The following example creates a semantic index named `ArticleIndex` on the textual documents in the ARTICLE column of the NEWSFEED table, using the extractor policy named `SEM_EXTR`:

```
CREATE INDEX ArticleIndex on Newsfeed (article)
   INDEXTYPE IS mdsys.SemContext PARAMETERS ('SEM_EXTR');
```

The RDF graph created for an index is managed internally and it is not associated with an application table. The triples stored in such an RDF graph are automatically maintained for any modifications (such as update, insert, or delete) made to the documents stored in the table column. Although a single RDF graph is used to index all documents stored in a table column, the triples stored in the RDF graph maintain references to the documents from which they are extracted; therefore, all the triples extracted from a specific document form an individual graph within the RDF graph. The documents that are semantically indexed can then be searched using a SPARQL query pattern that operates on the triples extracted from the documents.

When creating a semantic index for documents, you can use a basic extractor policy or a dependent policy, which may include one or more user-defined RDF graphs. When you create an index with a dependent extractor policy, the document search pattern specified using SPARQL could span the triples extracted from the documents as well as those defined in user-defined RDF graphs.

You can create an index using multiple extractor policies, in which case the triples extracted by the corresponding extractors are maintained separately in distinct RDF graphs. A document search query using one such index can select the specific policy to be used for answering the query. For example, an extractor policy named `CITY_EXTR` can be created to extract the names of the cities from a given document, and this extractor policy can be used in combination with the SEM_EXTR policy to create a semantic index, as in the following example:

```
CREATE INDEX ArticleIndex on Newsfeed (article)
   INDEXTYPE IS mdsys.SemContext PARAMETERS ('SEM_EXTR CITY_EXTR');
```

The first extractor policy in the PARAMETERS list is considered to be the default policy if a query does not refer to a specific policy; however, you can change the default extractor policy for a semantic index by using the SEM_RDFCTX.SET_DEFAULT_POLICY procedure, as in the following example:

```
begin
  sem_rdfctx.set_default_policy (index_name => 'ArticleIndex',
                                 policy_name => 'CITY_EXTR');
end;
/
```

5.4 SEM_CONTAINS and Ancillary Operators

You can use the SEM_CONTAINS operator in a standard SQL statement to search for documents or document references that are stored in relational tables.

This operator has the following syntax:

```
SEM_CONTAINS(
  column   VARCHAR2 / CLOB,
  sparql   VARCHAR2,
  policy   VARCHAR2,
  aliases  SEM_ALIASES,
  index_status  NUMBER,
  ancoper  NUMBER
 ) RETURN NUMBER;
```

The `column` and `sparql` attributes attribute are required. The other attributes are optional (that is, each can be a null value).

The `column` attribute identifies a VARCHAR2 or CLOB column in a relational table that stores the documents or references to documents that are semantically indexed. An index of type MDSYS.SEMCONTEXT must be defined in this column for the SEM_CONTAINS operator to use.

The `sparql` attribute is a string literal that defines the document search criteria, expressed in SPARQL format.

The optional `policy` attribute specifies the name of an extractor policy, usually to override the default policy. A semantic document index can have one or more extractor policies specified at index creation, and one of these policies is the default, which is used if the `policy` attribute is null in the call to SEM_CONTAINS.

The optional `aliases` attribute identifies one or more namespaces, including a default namespace, to be used for expansion of qualified names in the query pattern. Its data type is SEM_ALIASES, which has the following definition: `TABLE OF SEM_ALIAS`, where each SEM_ALIAS element identifies a namespace ID and namespace value. The SEM_ALIAS data type has the following definition: `(namespace_id VARCHAR2(30), namespace_val VARCHAR2(4000))`

The optional `index_status` attribute is relevant only when a dependent policy involving one or more inferred graphs is being used for the SEM_CONTAINS invocation. The `index_status` value identifies the minimum required validity status of the inferred graphs. The possible values are 0 (for VALID, the default), 1 (for INCOMPLETE), and 2 (for INVALID).

The optional ancoper attribute specifies a number as the binding to be used when the SEM_CONTAINS_SELECT ancillary operator is used with this operator in a query. The number specified for the ancoper attribute should be the same as number specified for the operbind attribute in the SEM_CONTAINS_SELECT ancillary operator.

The SEM_CONTAINS operator returns 1 for each document instance matching the specified search criteria, and returns 0 for all other cases.

For more information about using the SEM_CONTAINS operator, including an example, see Searching for Documents Using SPARQL Query Patterns.

- SEM_CONTAINS_SELECT Ancillary Operator
- SEM_CONTAINS_COUNT Ancillary Operator

5.4.1 SEM_CONTAINS_SELECT Ancillary Operator

You can use the SEM_CONTAINS_SELECT ancillary operator to return additional information about each document that matches some search criteria. This ancillary operator has a single numerical attribute (operbind) that associates an instance of the SEM_CONTAINS_SELECT ancillary operator with a SEM_CONTAINS operator by using the same value for the binding. This ancillary operator returns an object of type CLOB that contains the additional information from the matching document, formatted in SPARQL Query Results XML format.

The SEM_CONTAINS_SELECT ancillary operator has the following syntax:

```
SEM_CONTAINS_SELECT(
  operbind  NUMBER
 ) RETURN CLOB;
```

For more information about using the SEM_CONTAINS_SELECT ancillary operator, including examples, see Bindings for SPARQL Variables in Matching Subgraphs in a Document (SEM_CONTAINS_SELECT Ancillary Operator).

5.4.2 SEM_CONTAINS_COUNT Ancillary Operator

You can use the SEM_CONTAINS_COUNT ancillary operator for a SEM_CONTAINS operator invocation. For each matched document, it returns the count of matching subgraphs for the SPARQL graph pattern specified in the SEM_CONTAINS invocation.

The SEM_CONTAINS_COUNT ancillary operator has the following syntax:

```
SEM_CONTAINS_COUNT(
  operbind  NUMBER
 ) RETURN NUMBER;
```

The following example excerpt shows the use of the SEM_CONTAINS_COUNT ancillary operator to return the count of matching subgraphs for each matched document:

```
SELECT docId, SEM_CONTAINS_COUNT(1) as matching_subgraph_count
FROM   Newsfeed
WHERE  SEM_CONTAINS (article,
  '{ ?org   rdf:type         class:Organization  .
     ?org   pred:hasCategory  cat:BusinessFinance }', ..,
   1)= 1;
```

5.5 Searching for Documents Using SPARQL Query Patterns

Documents that are semantically indexed (that is, indexed using the mdsys.SemContext index type) can be searched using SEM_CONTAINS operator within a standard SQL query.

In the query, the SEM_CONTAINS operator must have at least two parameters, the first specifying the column in which the documents are stored and the second specifying the document search criteria expressed as a SPARQL query pattern, as in the following example:

```
SELECT docId FROM Newsfeed
WHERE  SEM_CONTAINS (article,
  '{ ?org  rdf:type  <http://www.example.com/classes/Organization>  .
     ?org  <http://example.com/pred/hasCategory>
             <http://www.example.com/category/BusinessFinance> }'
           )= 1;
```

The SPARQL query pattern specified with the SEM_CONTAINS operator is matched against the individual graphs corresponding to each document, and a document is considered to match a search criterion if the triples from the corresponding graph satisfy the query pattern. In the preceding example, the SPARQL query pattern identifies the individual graphs (thus, the documents) that refer to an `Organization` that belong to `BusinessFinance` category. The SQL query returns the rows corresponding to the matching documents in its result set. The preceding example assumes that the URIs used in the query are generated by the underlying extractor, and that you (the user searching for documents) are aware of the properties and terms that are generated by the extractor in use.

When you create an index using a dependent extractor policy that includes one or more user-defined RDF graphs, the triples asserted in the user RDF graphs are considered to be common to all the documents. Document searches involving such policies test the search criteria against the triples in individual graphs corresponding to the documents, combined with the triples in the user RDF graphs. For example, the following query identifies all articles referring to organizations in the state of New Hampshire, using the geographical ontology (`geo_ontology` RDF graph from a preceding example) that maps cities to states:

```
SELECT docId FROM   Newsfeed
WHERE  SEM_CONTAINS (article,
        '{ ?org     rdf:type          class:Organization  .
           ?org     pred:hasLocation  ?city .
           ?city    geo:hasState      state:NewHampshire }',
        'SEM_EXTR_PLUS_GEOONT',
               sem_aliases(
                  sem_alias('class', 'http://www.myorg.com/classes/'),
                  sem_alias('pred', 'http://www.myorg.com/pred/'),
                  sem_alias('geo', 'http://geoont.org/rel/'),
                  sem_alias('state', 'http://geoont.org/state/'))) = 1;
```

The preceding query, with a reference to the extractor policy SEM_EXTR_PLUS_GEOONT (created in an example in Extractor Policies), combines the triples extracted from the indexed documents and the triples in the user RDF graph to find matching documents. In this example, the name of the extractor policy is optional if the corresponding index is created with just this policy or if this is the default extractor policy for the index. When the query pattern uses some qualified names, an optional parameter to the SEM_CONTAINS operator can specify the namespaces to be used for expanding the qualified names.

SPARQL-based document searches can make use of the SPARQL syntax that is supported through SEM_MATCH queries.

5.6 Bindings for SPARQL Variables in Matching Subgraphs in a Document (SEM_CONTAINS_SELECT Ancillary Operator)

You can use the SEM_CONTAINS_SELECT ancillary operator to return additional information about each document matched using the SEM_CONTAINS operator.

Specifically, the bindings for the variables used in SPARQL-based document search criteria can be returned using this operator. This operator is ancillary to the SEM_CONTAINS operator, and a literal number is used as an argument to this operator to associate it with a specific instance of SEM_CONTAINS operator, as in the following example:

```
SELECT docId, SEM_CONTAINS_SELECT(1) as result
FROM   Newsfeed
WHERE  SEM_CONTAINS (article,
  '{ ?org   rdf:type          class:Organization  .
     ?org   pred:hasCategory  cat:BusinessFinance }', ..,
   1)= 1;
```

The SEM_CONTAINS_SELECT ancillary operator returns the bindings for the variables in SPARQL Query Results XML format, as CLOB data. The variables may be bound to multiple data instances from a single document, in which case all bindings for the variables are returned. The following example is an excerpt from the output of the preceding query: a value returned by the SEM_CONTAINS_SELECT ancillary operator for a document matching the specified search criteria.

```
<results>
  <result>
     <binding name="ORG">
        <uri>http://newscorp.com/Org/AcmeCorp</uri>
     </binding>
  </result>
  <result>
     <binding name="ORG">
       <uri>http://newscorp.com/Org/ABCCorp</uri>
     </binding>
  </result>
</results>
```

You can rank the search results by creating an instance of XMLType for the CLOB value returned by the SEM_CONTAINS_SELECT ancillary operator and applying an XPath expression to sort the results on some attribute values.

By default, the SEM_CONTAINS_SELECT ancillary operator returns bindings for all variables used in the SPARQL-based document search criteria. However, when the values for only a subset of the variables are relevant for a search, the SPARQL pattern can include a SELECT clause with space-separated list of variables for which the values should be returned, as in the following example:

```
SELECT docId, SEM_CONTAINS_SELECT(1) as result
FROM   Newsfeed
WHERE  SEM_CONTAINS (article,
        'SELECT ?org  ?city
         WHERE { ?org     rdf:type          class:Organization  .
                 ?org     pred:hasLocation  ?city .
                 ?city    geo:hasState      state:NewHampshire }', ..,
         1) = 1;
```

5.7 Improving the Quality of Document Search Operations

The quality of a document search operation depends on the quality of the information produced by the extractor used to index the documents. If the information extracted is incomplete, you may want to add some annotations to a document.

You can use the SEM_RDFCTX.MAINTAIN_TRIPLES procedure to add annotations, in the form of RDF triples, to specific documents in order to improve the quality of search, as shown in the following example:

```
begin
  sem_rdfctx.maintain_triples(
     index_name      => 'ArticleIndex',
     where_clause    => 'docid in (1,15,20)',
     rdfxml_content => sys.xmltype(
      '<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
                xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
                xmlns:pred="http://example.com/pred/">
       <rdf:Description rdf:about=" http://newscorp.com/Org/ExampleCorp">
         <pred:hasShortName
               rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
             Example
         </pred:hasShortName>
     </rdf:Description>
    </rdf:RDF>'));
end;
/
```

The index name and the WHERE clause specified in the preceding example identify specific instances of the document to be annotated, and the RDF/XML content passed in is used to add additional triples to the individual graphs corresponding to those documents. This allows domain experts and user communities to improve the quality of search by adding relevant triples to annotate some documents.

5.8 Indexing External Documents

You can use semantic indexing on documents that are stored in a file system or on the network. In such cases, you store the references to external documents in a table column, and you create a semantic index on the column using an appropriate extractor policy.

To index external documents, define an extractor policy with appropriate preferences, using an XML document that is assigned to the `preferences` parameter of the SEM_RDFCTX.CREATE_POLICY procedure, as in the following example:

```
begin
  sem_rdfctx.create_policy (
       policy_name => 'SEM_EXTR_FROM_FILE',
       extractor   => mdsys.gatenlp_extractor()),
       preferences => sys.xmltype('<RDFCTXPreferences>
                                     <Datastore type="FILE">
                                        <Path>EXTFILES_DIR</Path>
                                     </Datastore>
                                   </RDFCTXPreferences>'));
end;
/
```

The `<Datastore>` element in the preferences document specifies the type of repository used for the documents to be indexed. When the value for the `type` attribute is set to `FILE`, the

`<Path>` element identifies a directory object in the database (created using the SQL statement CREATE DIRECTORY). A table column indexed using the specified extractor policy is expected to contain relative paths to individual files within the directory object, as shown in the following example:

```
CREATE TABLE newsfeed (docid       number,
                       articleLoc  VARCHAR2(100));
INSERT INTO into newsfeed (docid, articleLoc) values
                     (1, 'article1.txt');
INSERT INTO newsfeed (docid, articleLoc) values
                     (2, 'folder/article2.txt');

CREATE INDEX ArticleIndex on newsfeed (articleLoc)
   INDEXTYPE IS mdsys.SemContext PARAMETERS ('SEM_EXTR_FROM_FILE');
```

To index documents that are accessed using HTTP protocol, create a extractor policy with preferences that set the `type` attribute of the `<Datastore>` element to `URL` and that list one or more hosts in the `<Path>` elements, as shown in the following excerpt:

```
<RDFCTXPreferences>
   <Datastore type="URL">
       <Path>http://cnn.com</Path>
       <Path>http://abc.com</Path>
   </Datastore>
</RDFCTXPreferences>
```

The schema in which a semantic index for external documents is created must have the necessary privileges to access the external objects, including access to any proxy server used to access documents outside the firewall, as shown in the following example:

```
-- Grant read access to the directory object for FILE data store --
grant read on directory EXTFILES_DIR to SEMUSR;

-- Grant connect access to set of hosts for URL data store --
begin
  dbms_network_acl_admin.create_acl (
                acl          => 'network_docs.xml',
                description  => 'Normal Access',
                principal    => 'SEMUSR',
                is_grant     => TRUE,
                privilege    => 'connect');
end;
/

begin
  dbms_network_acl_admin.assign_acl (
               acl        => 'network_docs.xml',
               host       =>  'cnn.com',
               lower_port => 1,
               upper_port => 10000);
end;
/
```

External documents that are semantically indexed in the database may be in one of the well-known formats such as Microsoft Word, RTF, and PDF. This takes advantage of the Oracle Text capability to extract plain text version from formatted documents using filters (see the CTX_DOC.POLICY_FILTER procedure, described in *Oracle Text Reference*). To semantically index formatted documents, you must specify the name of a CTX policy in the extractor preferences, as shown in the following excerpt:

```
<RDFCTXPreferences>
   <Datastore type="FILE" filter="CTX_FILTER_POLICY">
       <Path>EXTFILES_DIR</Path>
   </Datastore>
</RDFCTXPreferences>
```

In the preceding example, the `CTX_FILTER_POLICY` policy, created using the CTX_DDL.CREATE_POLICY procedure, must exist in your schema. The table columns that are semantically indexed using this preferences document can store paths to formatted documents, from which plain text is extracted using the specified CTX policy. The information extractor associated with the extractor policy then processes the plain text further, to extract the semantics in RDF/XML format.

5.9 Configuring the Calais Extractor type

The CALAIS_EXTRACTOR type, which is a subtype of the RDFCTX_WS_EXTRACTOR type, enables you to access a Web service end point anywhere on the network, including the one that is publicly accessible (`OpenCalais.com`).

To do so, you must connect as SYSTEM (not SYS … AS SYSDBA) or another non-SYS user with the DBA role, and configure the Calais extractor type with Web service end point, the SOAP action, and the license key by setting corresponding parameters, as shown in the following example:

```
begin
  sem_rdfctx.set_extractor_param (
     param_key   => 'CALAIS_WS_ENDPOINT',
     param_value => 'http://api1.opencalais.com/enlighten/calais.asmx',
     param_desc  => 'Calais web service end-point');

  sem_rdfctx.set_extractor_param (
     param_key   => 'CALAIS_KEY',
     param_value => '<Calais license key goes here>',
     param_desc  => 'Calais extractor license key');

  sem_rdfctx.set_extractor_param (
     param_key   => 'CALAIS_WS_SOAPACTION',
     param_value => 'http://clearforest.com/Enlighten',
     param_desc  => 'Calais web service SOAP Action');
end;
```

To enable access to a Web service outside the firewall, you must also set the parameter for the proxy host, as in the following example:

```
begin
  sem_rdfctx.set_extractor_param (
      param_key   => 'HTTP_PROXY',
      param_value => 'www-proxy.example.com',
      param_desc  => 'Proxy server');
end;
```

5.10 Working with General Architecture for Text Engineering (GATE)

General Architecture for Text Engineering (GATE) is an open source natural language processor and information extractor.

For details about GATE, see `http://gate.ac.uk`.

You can use GATE to perform semantic indexing of documents stored in the database. The extractor type `mdsys.gatenlp_extractor` is defined as a subtype of the RDFCTX_EXTRACTOR type. The implementation of this extractor type sends an unstructured document to a GATE engine over a TCP connection, receives corresponding annotations, and converts them into RDF following a user-specified XML style sheet.

The requests for information extraction are handled by a server socket implementation, which instantiates the GATE components and listens to extraction requests at a pre-determined port. The host and the post for the GATE listener are recorded in the database, as shown in the following example, for all instances of the `mdsys.gatenlp_extractor` type to use.

```
begin
  sem_rdfctx.set_extractor_param (
     param_key   => 'GATE_NLP_HOST',
     param_value => 'gateserver.example.com',
     param_desc  => 'Host for GATE NLP Listener ');

  sem_rdfctx.set_extractor_param (
     param_key   => 'GATE_NLP_PORT',
     param_value => '7687',
     param_desc  => 'Port for Gate NLP Listener');
end;
```

The server socket application receives an unstructured document and constructs an annotation set with the desired types of annotations. Each annotation in the set may be customized to include additional features, such as the relevant phrase from the input document and some domain specific features. The resulting annotation set is serialized into XML (using the `annotationSetToXml` method in the `gate.corpora.DocumentXmlUtils` Java package) and returned back to the socket client.

A sample Java implementation for the GATE listener is available for download from the code samples and examples page on OTN (see RDF Graph Management Examples (PL/SQL and Java) for information about this page).

The `mdsys.gatenlp_extractor` implementation in the database receives the annotation set encoded in XML, and converts it to RDF/XML using an XML style sheet. You can replace the default style sheet (listed in Default Style Sheet for GATE Extractor Output) used by the `mdsys.gatenlp_extractor` implementation with a custom style sheet when you instantiate the type.

The following example creates an extractor policy that uses a custom style sheet to generate RDF from the annotation set produced by the GATE extractor:

```
begin
  sem_rdfctx.create_policy (policy_name => 'GATE_EXTR',
                          extractor   => mdsys.gatenlp_extractor(
      sys.XMLType('<?xml version="1.0"?>
                 <xsl:stylesheet version="2.0"
                    xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >
                   ..
                 </xsl:stylesheet>')));
end;
/
```

5.11 Creating a New Extractor Type

You can create a new extractor type by extending the RDFCTX_EXTRACTOR or RDFCTX_WS_EXTRACTOR extractor type.

The extractor type to be extended must be accessible using Web service calls. The schema in which the new extractor type is created must be granted additional privileges to allow creation of the subtype. For example, if a new extractor type is created in the schema RDFCTXU, you must enter the following commands to grant the UNDER and RDFCTX_ADMIN privileges to that schema:

```
GRANT under ON mdsys.rdfctx_extractor TO rdfctxu;
GRANT rdfctx_admin TO rdfctxu;
```

As an example, assume that an information extractor can process an incoming document and return an XML document that contains extracted information. To enable the information extractor to be invoked using a PL/SQL wrapper, you can create the corresponding extractor type implementation, as in the following example:

```
create or replace type rdfctxu.info_extractor under rdfctx_extractor (
  xsl_trans   sys.XMLtype,
  constructor function info_extractor (
                 xsl_trans  sys.XMLType ) return self as result,
  overriding member function getDescription return VARCHAR2,
  overriding member function rdfReturnType return VARCHAR2,
  overriding member function extractRDF(document CLOB,
                                        docId    VARCHAR2) return CLOB
)
/

create or replace type body rdfctxu.info_extractor as
  constructor function info_extractor (
                 xsl_trans  sys.XMLType ) return self as result is
  begin
    self.extr_type := 'Info Extractor Inc.';
    -- XML style sheet to generate RDF/XML from proprietary XML documents
    self.xsl_trans := xsl_trans;
    return;
  end info_extractor;

  overriding member function getDescription return VARCHAR2 is
  begin
    return 'Extactor by Info Extractor Inc.';
  end getDescription;

  overriding member function rdfReturnType return VARCHAR2 is
  begin
    return 'RDF/XML';
  end rdfReturnType;

  overriding member function extractRDF(document CLOB,
                                        docId    VARCHAR2) return CLOB is
    ce_xmlt  sys.xmltype;
  begin
    EXECUTE IMMEDIATE
      'begin :1 = info_extract_xml(doc => :2); end;'
       USING IN OUT ce_xmlt, IN document;

    -- Now pass the ce_xmlt through RDF/XML transformation --
    return ce_xmlt.transform(self.xsl_trans).getClobVal();
  end extractRdf;

end;
```

In the preceding example:

- The implementation for the created `info_extractor` extractor type relies on the XML style sheet, set in the constructor, to generate RDF/XML from the proprietary XML schema used by the underlying information extractor.
- The `extractRDF` function assumes that the `info_extract_xml` function contacts the desired information extractor and returns an XML document with the information extracted from the document that was passed in.
- The XML style sheet is applied on the XML document to generate equivalent RDF/XML, which is returned by the `extractRDF` function.

5.12 Creating a Local Semantic Index on a Range-Partitioned Table

A local index can be created on a VARCHAR2 or CLOB column of a range-partitioned table.

To do so, use the following syntax:

```
CREATE INDEX <index-name> … LOCAL;
```

The following example creates a range-partitioned table and a local semantic index on that table:

```
CREATE TABLE part_newsfeed (
  docid number, article CLOB, cdate DATE)
partition by range (cdate)
(partition p1 values less than (to_date('01-Jan-2001')),
 partition p2 values less than (to_date('01-Jan-2004')),
 partition p3 values less than (to_date('01-Jan-2008')),
 partition p4 values less than (to_date('01-Jan-2012'))
);

CREATE INDEX ArticleLocalIndex on part_newsfeed (article)
   INDEXTYPE IS mdsys.SemContext PARAMETERS ('SEM_EXTR')
LOCAL;
```

Note that every partition of the local semantic index will have content generated for the same set of policies. When you use the ALTER INDEX statement on a local index to add or drop policies associated with a semantic index partition, you should try to keep the same set of policies associated with each partition. You can achieve this result by using ALTER INDEX statements in a loop over the set of partitions. (For more information about altering semantic indexes, see Altering a Semantic Index,)

5.13 Altering a Semantic Index

You can use the ALTER INDEX statement with a semantic index.

For a local semantic index, the ALTER INDEX statement applies to a specified partition. The general syntax of the ALTER INDEX command for a semantic index is as follows:

```
ALTER INDEX <index-name> REBUILD [PARTITION <index-partition-name>]
  [PARAMETERS ('-<action_for_policy> <policy-name>')];
```

- Rebuilding Content for All Existing Policies in a Semantic Index
- Rebuilding to Add Content for a New Policy to a Semantic Index
- Rebuilding Content for an Existing Policy from a Semantic Index

- Rebuilding to Drop Content for an Existing Policy from a Semantic Index

5.13.1 Rebuilding Content for All Existing Policies in a Semantic Index

If the PARAMETERS clause is not included in the ALTER INDEX statement, the content of the semantic index (or index partition) is rebuilt for every policy presently associated with the index. The following are two examples:

```
ALTER INDEX ArticleIndex REBUILD;
ALTER INDEX ArticleLocalIndex REBUILD PARTITION p1;
```

5.13.2 Rebuilding to Add Content for a New Policy to a Semantic Index

Using `add_policy` for <*action_for_policy*>, you can add content for a new base policy or a dependent policy to a semantic index (or index partition). If a dependent policy is being added and if its base policy is not already a part of the index, then content for the base policy is also added implicitly (by invoking the extractor specified as part of the base policy definition). The following is an example:

```
ALTER INDEX ArticleIndex REBUILD PARAMETERS ('-add_policy MY_POLICY');
```

5.13.3 Rebuilding Content for an Existing Policy from a Semantic Index

Using `rebuild_policy` for <*action_for_policy*>, you can rebuild the content of the semantic index (or index partition) for an existing policy presently associated with the index. The following is an example:

```
ALTER INDEX ArticleIndex REBUILD PARAMETERS ('-rebuild_policy MY_POLICY');
```

5.13.4 Rebuilding to Drop Content for an Existing Policy from a Semantic Index

Using `drop_policy` for <*action_for_policy*>, you can drop content corresponding to an existing base policy or a dependent policy from a semantic index (or index partition). Note that dropping the content for a base policy will fail if it is the only policy for the index (or index partition) or if it is used by dependent policies associated with this index (or index partition).

The following example drops the content for a policy from an index:

```
ALTER INDEX ArticleIndex REBUILD PARAMETERS ('-drop_policy MY_POLICY');
```

5.14 Passing Extractor-Specific Parameters in CREATE INDEX and ALTER INDEX

The CREATE INDEX and ALTER INDEX statements allow the passing of parameters needed by extractors.

These parameters are passed on to the extractor using the `params` parameter of the `extractRdf` and `batchExtractRdf` methods. The following two examples show their use:

```
CREATE INDEX ArticleIndex on Newsfeed (article)
  INDEXTYPE IS mdsys.SemContext PARAMETERS ('SEM_EXTR=(NE_ONLY)');

ALTER INDEX ArticleIndex REBUILD
  PARAMETERS ('-add_policy MY_POLICY=(NE_ONLY)');
```

5.15 Performing Document-Centric Inference

Document-centric inference refers to the ability to infer from each document individually.

It does not allow triples extracted from two different documents to be used together for inference. It contrasts with the more common corpus-centric inference, where new triples can be inferred from combinations of triples extracted from multiple documents.

Document-centric inference can be desirable in document search applications because inclusion of a document in the search result is based on the extracted and/or inferred triples for that document only, that is, triples extracted and/or inferred from any other documents in the corpus do not play any role in the selection of this document. (Document-centric inference might be preferred, for example, if there is inconsistency among documents because of differences in the reliability of the data or in the biases of the document creators.)

To perform document-centric inference, use named graph based local inference (explained in Named Graph Based Local Inference (NGLI)) by specifying `options => 'LOCAL_NG_INF=T'` in the call to the SEM_APIS.CREATE_INFERRED_GRAPH procedure.

Inferred graphs created through document-centric inference can be included as content of a semantic index by creating a dependent policy and adding that policy to the semantic index, as shown in Example 5-2.

Example 5-2 Using Document-Centric Inference

```
-- Create inferred graph 'extr_data_inf' using document-centric inference
-- assuming:
--   model_name for semantic index based on base policy: 'RDFCTX_MOD_1'
--    (model name (RDF graph name)is available from the RDFCTX_INDEX_POLICIES view;
--     see RDFCTX_INDEX_POLICIES View)
--   ontology: dataOntology
--   rulebase: OWL2RL
-- options: 'LOCAL_NG_INF=T' (for document-centric inference)
BEGIN
sem_apis.create_inferred_graph('extr_data_inf',
  models_in    => sem_models('RDFCTX_MOD_1', 'dataOntology'),
  rulebases_in => sem_rulebases('OWL2RL'),
  options      => 'LOCAL_NG_INF=T');
END;
/
-- Create a dependent policy to augment data extracted using base policy
-- with content of inferred graph extr_data_inf (computed in previous statement)
BEGIN
sem_rdfctx.create_policy (
  policy_name => 'SEM_EXTR_PLUS_DATA_INF',
  base_policy => 'SEM_EXTR',
  user_models => NULL,
  user_entailments => sem_models('extr_data_inf'));
END;
/
-- Add the dependent policy to the ARTICLEINDEX index.
EXECUTE sem_rdfctx.add_dependent_policy('ARTICLEINDEX','SEM_EXTR_PLUS_DATA_INF');
```

5.16 Metadata Views for Semantic Indexing

This section describes views that contain metadata about semantic indexing

- RDFCTX_POLICIES View

- RDFCTX_INDEX_POLICIES View
- RDFCTX_INDEX_EXCEPTIONS View

5.16.1 RDFCTX_POLICIES View

Information about extractor policies defined in the current schema is maintained in the RDFCTX_POLICIES view, which has the columns shown in Table 5-1 and one row for each extractor policy.

Table 5-1 RDFCTX_POLICIES View Columns

Column Name	Data Type	Description
POLICY_OWNER	VARCHAR2(32)	Owner of the extractor policy
POLICY_NAME	VARCHAR2(32)	Name of the extractor policy
EXTRACTOR	MDSYS.RDFCTX_EXTRACTOR	Instance of extractor type
IS_DEPENDENT	VARCHAR2(3)	Contains `YES` if the extractor policy is dependent on a base policy; contains `NO` if the extractor policy is not dependent on a base policy.
BASE_POLICY	VARCHAR2(32)	For a dependent policy, the name of the base policy
USER_MODELS	SEM_MODELS	For a dependent policy, a list of the RDF graphs included in the policy

5.16.2 RDFCTX_INDEX_POLICIES View

Information about semantic indexes defined in the current schema and the extractor policies used to create the index is maintained in the RDFCTX_POLICIES view, which has the columns shown in Table 5-2 and one row for each combination of semantic index and extractor policy.

Table 5-2 RDFCTX_INDEX_POLICIES View Columns

Column Name	Data Type	Description
INDEX_OWNER	VARCHAR2(32)	Owner of the semantic index
INDEX_NAME	VARCHAR2(32)	Name of the semantic index
INDEX_PARTITION	VARCHAR2(32)	Name of the index partition (for LOCAL index only)
POLICY_NAME	VARCHAR2(32)	Name of the extractor policy
EXTR_PARAMETERS	VARCHAR2(100)	Parameters specified for the extractor
IS_DEFAULT	VARCHAR2(3)	Contains `YES` if POLICY_NAME is the default extractor policy for the index; contains `NO` if POLICY_NAME is not the default extractor policy for the index.

Table 5-2 (Cont.) RDFCTX_INDEX_POLICIES View Columns

Column Name	Data Type	Description
STATUS	VARCHAR2(10)	Contains `VALID` if the index is valid, `INPROGRESS` if the index is being created, or `FAILED` if a system failure occurred during the creation of the index.
RDF_MODEL	VARCHAR2(32)	Name of the RDF graph maintaining the index data

5.16.3 RDFCTX_INDEX_EXCEPTIONS View

Information about exceptions encountered while creating or maintaining semantic indexes in the current schema is maintained in the RDFCTX_INDEX_EXCEPTIONS view, which has the columns shown in Table 5-3 and one row for each exception.

Table 5-3 RDFCTX_INDEX_EXCEPTIONS View Columns

Column Name	Data Type	Description
INDEX_OWNER	VARCHAR2(32)	Owner of the semantic index associated with the exception
INDEX_NAME	VARCHAR2(32)	Name of the semantic index associated with the exception
POLICY_NAME	VARCHAR2(32)	Name of the extractor policy associated with the exception
DOC_IDENTIFIER	VARCHAR2(38)	Row identifier (rowid) of the document associated with the exception
EXCEPTION_TYPE	VARCHAR2(13)	Type of exception
EXCEPTION_CODE	NUMBER	Error code associated with the exception
EXCEPTION_TEXT	CLOB	Text associated with the exception
EXTRACTED_AT	TIMESTAMP	Time at which the exception occurred

5.17 Default Style Sheet for GATE Extractor Output

This section lists the default XML style sheet that the `mdsys.gatenlp_extractor` implementation uses to convert the annotation set (encoded in XML) into RDF/XML.

(This extractor is explained in Working with General Architecture for Text Engineering (GATE).)

```
<?xml version="1.0"?>
  <xsl:stylesheet version="2.0"
                 xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >
     <xsl:output encoding="utf-8" indent="yes"/>
     <xsl:param name="docbase">http://xmlns.oracle.com/rdfctx/</xsl:param>
     <xsl:param name="docident">0</xsl:param>
     <xsl:param name="classpfx">
```

```
    <xsl:value-of select="$docbase"/>
    <xsl:text>class/</xsl:text>
  </xsl:param>
  <xsl:template match="/">
     <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
              xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
              xmlns:owl="http://www.w3.org/2002/07/owl#"
              xmlns:prop="http://xmlns.oracle.com/rdfctx/property/">
     <xsl:for-each select="AnnotationSet/Annotation">
       <rdf:Description>
         <xsl:attribute name="rdf:about">
           <xsl:value-of select="$docbase"/>
           <xsl:text>docref/</xsl:text>
           <xsl:value-of select="$docident"/>
           <xsl:text>/</xsl:text>
           <xsl:value-of select="@Id"/>
         </xsl:attribute>
         <xsl:for-each select="./Feature">
           <xsl:choose>
             <xsl:when test="./Name[text()='majorType']">
               <rdf:type>
                 <xsl:attribute name="rdf:resource">
                    <xsl:value-of select="$classpfx"/>
                    <xsl:text>major/</xsl:text>
                    <xsl:value-of select="translate(./Value/text(),
                                                    ' ', '#')"/>
                 </xsl:attribute>
               </rdf:type>
             </xsl:when>
             <xsl:when test="./Name[text()='minorType']">
               <xsl:element name="prop:hasMinorType">
                 <xsl:attribute name="rdf:resource">
                    <xsl:value-of select="$docbase"/>
                    <xsl:text>minorType/</xsl:text>
                    <xsl:value-of select="translate(./Value/text(),
                                                    ' ', '#')"/>
                 </xsl:attribute>
               </xsl:element>
             </xsl:when>
             <xsl:when test="./Name[text()='kind']">
               <xsl:element name="prop:hasKind">
                 <xsl:attribute name="rdf:resource">
                    <xsl:value-of select="$docbase"/>
                    <xsl:text>kind/</xsl:text>
                    <xsl:value-of select="translate(./Value/text(),
                                                    ' ', '#')"/>
                 </xsl:attribute>
               </xsl:element>
             </xsl:when>
             <xsl:when test="./Name[text()='locType']">
               <xsl:element name="prop:hasLocType">
                 <xsl:attribute name="rdf:resource">
                    <xsl:value-of select="$docbase"/>
                    <xsl:text>locType/</xsl:text>
                    <xsl:value-of select="translate(./Value/text(),
                                                    ' ', '#')"/>
                 </xsl:attribute>
               </xsl:element>
             </xsl:when>
             <xsl:when test="./Name[text()='entityValue']">
               <xsl:element name="prop:hasEntityValue">
                 <xsl:attribute name="rdf:datatype">
```

```
                <xsl:text>
                   http://www.w3.org/2001/XMLSchema#string
                </xsl:text>
              </xsl:attribute>
              <xsl:value-of select="./Value/text()"/>
            </xsl:element>
          </xsl:when>
          <xsl:otherwise>
            <xsl:element name="prop:has{translate(
                                  substring(./Name/text(),1,1),
                                  'abcdefghijklmnopqrstuvwxyz',
                                  'ABCDEFGHIJKLMNOPQRSTUVWXYZ')}{
                                substring(./Name/text(),2)}">
               <xsl:attribute name="rdf:datatype">
                  <xsl:text>
                    http://www.w3.org/2001/XMLSchema#string
                  </xsl:text>
               </xsl:attribute>
              <xsl:value-of select="./Value/text()"/>
            </xsl:element>
          </xsl:otherwise>
        </xsl:choose>
      </xsl:for-each>
    </rdf:Description>
  </xsl:for-each>
  </rdf:RDF>
</xsl:template>
</xsl:stylesheet>
```

6 Fine-Grained Access Control for RDF Data

The default control of access to the Oracle Database RDF data store is at the RDF graph level: the owner of a graph can grant select, delete, and insert privileges on the graph to other users by granting appropriate privileges on the view named RDFM_*<rdf_graph_name>*. However, for applications with stringent security requirements, you can enforce a fine-grained access control mechanism by using the Oracle Label Security option of Oracle Database.

Oracle Label Security (OLS) for RDF data allows sensitivity labels to be associated with individual triples stored in an RDF graph. For each query, access to specific triples is granted by comparing their labels with the user's session labels. This triple-level security option provides a thin layer of RDF-specific capabilities on top of the Oracle Database native support for label security.

For information about using OLS, see *Oracle Label Security Administrator's Guide*.

- Triple-Level Security
  The triple-level security option provides a thin layer of RDF-specific capabilities on top of the Oracle Database native support for label security.

6.1 Triple-Level Security

The triple-level security option provides a thin layer of RDF-specific capabilities on top of the Oracle Database native support for label security.

To use triple-level security, specify `SEM_RDFSA.TRIPLE_LEVEL_ONLY` as the `rdfsa_options` parameter value when you execute the SEM_RDFSA.APPLY_OLS_POLICY procedure. For example:

```
EXECUTE sem_rdfsa.apply_ols_policy('defense', SEM_RDFSA.TRIPLE_LEVEL_ONLY,
network_owner=>'FGAC_ADMIN', network_name=>'OLS_NET');
```

Do not specify any of the other available parameters for the SEM_RDFSA.APPLY_OLS_POLICY procedure.

When you use triple-level security, OLS is applied to each RDF graph in the network. That is, label security is applied to the relevant internal tables and to all the application tables; there is no need to manually apply policies to the application tables of existing RDF graphs. However, if you need to create additional graphs after applying the OLS policy, you must use the SEM_OLS.APPLY_POLICY_TO_APP_TAB procedure to apply OLS to the application table before creating the RDF graph. Similarly, if you have dropped an RDF graph and you no longer need to protect the application table, you can use the SEM_OLS.REMOVE_POLICY_FROM_APP_TAB procedure. (These procedures are described in SEM_OLS Package Subprograms.)

With triple-level security, duplicate triples with different labels can be inserted in the RDF graph. (Such duplicates are not allowed with resource-level security.) For example, assume that you have a triple with a very sensitive label, such as:

```
(<urn:X>,<urn:P>,<urn:Y>, "TOPSECRET")
```

This does not prevent a low-privileged (`UNCLASSIFIED`) user from inserting the triple `(<urn:X>,<urn:P>,<urn:Y>, "UNCLASSIFIED")`. Because SPARQL and SEM_MATCH do not return label information, a query will return both rows (assuming the user has appropriate privileges), and it will not be easy to distinguish between the `TOPSECRET` and `UNCLASSIFIED` triples.

To filter out such low-security triples when querying the RDF graphs, you can one or more the following options with SEM_MATCH:

- `POLICY_NAME` specifies the OLS policy name.
- `MIN_LABEL` specifies the minimum label for triples that are included in the query

In other words, every triple that contains a label that is strictly dominated by `MIN_LABEL` is not included in the query. For example, to filter out the "UNCLASSIFIED" triple, you could use the following query (assuming the OLS policy name is `DEFENSE` and that the query user has read privileges over `UNCLASSIFIED` and `TOPSECRET` triples):

```
SELECT s,p,y FROM table(sem_match('{?s ?p ?y}' ,
  sem_models(TEST'), null, null, null, null,
  'MIN_LABEL=TOPSECRET POLICY_NAME=DEFENSE',
  null, null, 'FGAC_ADMIN', 'OLS_NET'));
```

Note that the filtering in the preceding example occurs in addition to the security checks performed by the native OLS software.

After a triple has been inserted, you can view and update the label information through the `CTXT1` column in the application table for the RDF graph (assuming that you have the `WRITEUP` and `WRITEDOWN` privileges to modify the labels).

There are no restrictions on who can perform inference or bulk loading with triple-level security; all of the inferred or bulk loaded triples are inserted with the user's session row label. Note that you can change the session labels by using the SA_UTL package. (For more information about SA_UTL, see *Oracle Label Security Administrator's Guide*.)

- Fine-Grained Security for Inferred Data and Ladder-Based Inference (LBI)
- Extended Example: Applying OLS Triple-Level Security on RDF Data

6.1.1 Fine-Grained Security for Inferred Data and Ladder-Based Inference (LBI)

When triple-level security is turned on for RDF data stored in Oracle Database, asserted facts are tagged with data labels to enforce mandatory access control. In addition, when a user invokes the forward-chaining based inference function through the SEM_APIS.CREATE_INFERRED_GRAPH procedure, the newly inferred relationships will be tagged with the current row label (`SA_UTL.NUMERIC_ROW_LABEL`).

These newly inferred relationships are derived solely based on the information that the user is allowed to access. These relationships do, however, share the same data label. This is understandable because a SEM_APIS.CREATE_INFERRED_GRAPH call can be viewed as a three-step process: read operation, followed by a logical inference computation, followed by a write operation. The read operation gathers information upon which inference computation is based, and it is restricted by access privileges, the user's label, and the data labels; the logical inference computation step is purely mathematical; and the final write of inferred information into the entailed graph is no different from the same user asserting some new facts (which happen to be calculated by the previous step).

Having all inferred assertions tagged with a single label is sufficient if a user only owns a single label. It is, however, not fine-grained enough when there are multiple labels owned by the same user, which is a common situation in a multitenancy setup.

For example, assume a user sets its user label and data label as `TopSecret`, invokes SEM_APIS.CREATE_INFERRED_GRAPH, switches to a weaker label named `Secret`, and finally performs a SPARQL query. The query will not be able to see any of those newly inferred relationships because they were all tagged with the `TopSecret` label. However, if the user switches back to the `TopSecret` label, now every single inferred relationship is visible. It is "all or nothing" (that is, all visible or nothing visible) as far as inferred relationships are concerned.

When multiple labels are available for use by a given user, you normally want to assign different labels to different inferred relationships. There are two ways to achieve this goal:

- Invoking SEM_APIS.CREATE_INFERRED_GRAPH Multiple Times
- Using Ladder-Based Inference (LBI)

Ladder-based inference, effective with Oracle Database 12*c* Release 1 (12.1), is probably the simpler and more convenient of the two approaches.

Invoking SEM_APIS.CREATE_INFERRED_GRAPH Multiple Times

Assume a security policy named DEFENSE, a user named SCOTT, and a sequence of user labels Label1, Label2,..., Label*n* owned by SCOTT. The following call by SCOTT sets the label as Label1, runs the inference for the first time, and tags the newly inferred triples with Label1:

```
EXECUTE sa_utl.set_label('defense',char_to_label('defense','Label1'));
EXECUTE sa_utl.set_row_label('defense',char_to_label('defense','Label1'));
EXECUTE sem_apis.create_inferred_graph('inf', sem_models('contracts'),
sem_rulebases('owlprime'), SEM_APIS.REACH_CLOSURE,
null,'',network_owner=>'FGAC_ADMIN',network_name=>'OLS_NET');
```

Now, SCOTT switches the label to Label2, runs the inference a second time, and tags the newly inferred triples with Label2. Obviously, if Label2 is dominated by Label1, then no new triples will be inferred because Label2 cannot see anything beyond what Label1 is allowed to see. If Label2 is not dominated by Label1, the read step of the inference process will probably see a different set of triples, and consequently the inference call can produce some new triples, which will in turn be tagged with Label2.

For the purpose of this example, assume the following condition holds true: for any 1 <= *i* < *j* <= *n*, Label*j* is not dominated by Label*i*.

```
EXECUTE sa_utl.set_label('defense',char_to_label('defense','Label2'));
EXECUTE sa_utl.set_row_label('defense',char_to_label('defense','Label2'));
EXECUTE sem_apis.create_inferred_graph('inf', sem_models('contracts'),
sem_rulebases('owlprime'), SEM_APIS.REACH_CLOSURE, null, 'ENTAIL_ANYWAY=T',
network_owner=>'FGAC_ADMIN', network_name=>'OLS_NET');
```

SCOTT continues the preceding actions using the rest of the labels in the label sequence: Label1, Label2, ..., Label*n*. The last step will be as follows:

```
EXECUTE sa_utl.set_label('defense',char_to_label('defense','Labeln'));
EXECUTE sa_utl.set_row_label('defense',char_to_label('defense','Labeln'));
EXECUTE sem_apis.create_inferred_graph('inf', sem_models('contracts'),
sem_rulebases('owlprime'), SEM_APIS.REACH_CLOSURE, null, 'ENTAIL_ANYWAY=T',
network_owner=>'FGAC_ADMIN', network_name=>'OLS_NET');
```

After all these actions are performed, the inference graph probably consists of triples tagged with various different labels.

Using Ladder-Based Inference (LBI)

Basically, ladder-based inference (LBI) wraps in one API call all the actions described in the Invoking SEM_APIS.CREATE_INFERRED_GRAPH Multiple Times approach. Visually, those actions are like climbing up a ladder. When proceeding from one label to the next, more asserted facts become visible or accessible (assuming the new label is not dominated by any of the previous ones), and therefore new relationships can be inferred.

The syntax to invoke LBI is shown in the following example.

```
EXECUTE sem_apis.create_inferred_graph('inf',
  sem_models('contracts'),
  sem_rulebases('owlprime'),
  SEM_APIS.REACH_CLOSURE,
  null,
  null,
  ols_ladder_inf_lbl_seq=>'numericLabel1 numericLabel2 numericLabel3 numericLabel4',
  network_owner=>'FGAC_ADMIN',
  network_name=>'OLS_NET'
);
```

The parameter `ols_ladder_inf_lbl_seq` specifies a sequence of labels. This sequence is provided as a list of numeric labels delimited by spaces. When using LBI, it is a good practice to arrange the sequence of labels so that weaker labels are put before stronger labels. This will reduce the size of the inferred graph. (If labels do not dominate each other, they can be specified in any order.)

6.1.2 Extended Example: Applying OLS Triple-Level Security on RDF Data

This section presents an extended example illustrating how to apply OLS triple-level security to RDF data. It assumes that OLS has been configured and enabled. The examples are very simplified, and do not reflect recommended practices regarding user names and passwords.

Unless otherwise indicated, perform the steps while connected AS SYSDBA.

1. Perform some necessary setup steps.
   a. As SYSDBA, create database users named A, B, and C.

      ```
      create user a identified by <password-for-a>;
      grant connect, unlimited tablespace, resource to a;
      create user b identified by <password-for-b>;
      grant connect, unlimited tablespace, resource to b;
      create user c identified by <password-for-c>;
      grant connect, unlimited tablespace, resource to c;
      ```

   b. As SYSDBA, create a security administrator and grant privileges.

      ```
      CREATE USER fgac_admin identified by <password-for-fgac_admin>;
      GRANT connect, unlimited tablespace,resource to fgac_admin;

      -- Needed to administer OLS on a shared schema-private network
      GRANT execute on MDSYS.SEM_RDFSA to fgac_admin;
      GRANT exempt access policy to fgac_admin;

      -- Needed to administer an OLS policy
      GRANT EXECUTE ON sa_components TO fgac_admin;
      GRANT EXECUTE ON sa_user_admin TO fgac_admin;
      GRANT EXECUTE ON sa_label_admin TO fgac_admin;
      GRANT EXECUTE ON sa_policy_admin TO fgac_admin;
      GRANT EXECUTE ON sa_sysdba to fgac_admin;
      GRANT EXECUTE ON TO_LBAC_DATA_LABEL to fgac_admin;
      GRANT lbac_dba to fgac_admin;
      ```

   c. Connect as SYSTEM and create a schema-private RDF network owned by the security administrator with sharing privileges.

```
CONNECT system/<password-for-system>;
EXECUTE
sem_apis.create_sem_network('tbs_3',network_owner=>'FGAC_ADMIN',network_name=>'OL
S_NET');
EXECUTE sem_apis.grant_network_sharing_privs('FGAC_ADMIN');
```

d. Connect as the security administrator and set up network sharing for users a, b, and c.

```
CONNECT fgac_admin/<password-for- fgac_admin>;
EXECUTE
sem_apis.enable_network_sharing(network_owner=>'FGAC_ADMIN',network_name=>'OLS_NE
T');
EXECUTE
sem_apis.grant_network_access_privs(network_owner=>'FGAC_ADMIN',network_name=>'OL
S_NET', network_user=>'A');
EXECUTE
sem_apis.grant_network_access_privs(network_owner=>'FGAC_ADMIN',network_name=>'OL
S_NET', network_user=>'B');
EXECUTE
sem_apis.grant_network_access_privs(network_owner=>'FGAC_ADMIN',network_name=>'OL
S_NET', network_user=>'C');
```

e. Connect as the security administrator and create a policy named defense.

```
CONNECT fgac_admin/<password-for-fgac_admin>;
EXECUTE SA_SYSDBA.CREATE_POLICY('defense','ctxt1');
```

f. Create three security levels (For simplicity, compartments and groups are omitted.)

```
EXECUTE SA_COMPONENTS.CREATE_LEVEL('defense',3000,'TS','TOP SECRET');
EXECUTE SA_COMPONENTS.CREATE_LEVEL('defense',2000,'SE','SECRET');
EXECUTE SA_COMPONENTS.CREATE_LEVEL('defense',1000,'UN','UNCLASSIFIED');
```

g. Create three labels.

```
EXECUTE SA_LABEL_ADMIN.CREATE_LABEL('defense',1000,'UN');
EXECUTE SA_LABEL_ADMIN.CREATE_LABEL('defense',1500,'SE');
EXECUTE SA_LABEL_ADMIN.CREATE_LABEL('defense',3100,'TS');
```

h. Assign labels and privileges.

```
EXECUTE SA_USER_ADMIN.SET_USER_LABELS('defense','A','UN');
EXECUTE SA_USER_ADMIN.SET_USER_LABELS('defense','B','SE');
EXECUTE SA_USER_ADMIN.SET_USER_LABELS('defense','C','TS');
EXECUTE SA_USER_ADMIN.SET_USER_LABELS('defense','fgac_admin','TS');
EXECUTE SA_USER_ADMIN.SET_USER_PRIVS('defense','FGAC_ADMIN', 'full');
```

2. Create an RDF graph.

a. Create an RDF graph and share it with some other users.

```
CONNECT a/<password-for-a>
CREATE TABLE project_tpl (triple sdo_rdf_triple_s) compress for oltp;
EXECUTE sem_apis.create_rdf_graph('project', 'project_tpl',
'triple' ,network_owner=>'FGAC_ADMIN',network_name=>'OLS_NET');
GRANT select on fgac_admin.ols_net#rdfm_project to B;
GRANT select on fgac_admin.ols_net#rdfm_project to C;
GRANT select, insert, update, delete on project_tpl to B, C;
```

b. Ensure that the bulk loading API can be executed.

```
GRANT insert on project_tpl to fgac_admin;
```

3. Apply the OLS policy for RDF.

```
CONNECT fgac_admin/<password-for-fgac_admin>
BEGIN
  sem_rdfsa.apply_ols_policy('defense',
```

```
sem_rdfsa.TRIPLE_LEVEL_ONLY,network_owner=>'FGAC_ADMIN',network_name=>'OLS_NET');
END;
/

/
```

Note that the application table now has an extra column named CTXT1:

```
CONNECT a/<password-for-a>
DESCRIBE project_tpl;
Name                                      Null?    Type
----------------------------------------- -------- --------------------------
 TRIPLE                                             PUBLIC.SDO_RDF_TRIPLE_S
 CTXT1                                              NUMBER(10)
```

4. Add data to the RDF graph.

```
-- User A uses incremental APIs to add RDF data
connect a/<password-for-a>
INSERT INTO project_tpl(triple) values
(sdo_rdf_triple_s('project','<urn:A>','<urn:hasManager>','<urn:B>','FGAC_ADMIN','OLS_
NET'));
INSERT INTO project_tpl(triple) values
(sdo_rdf_triple_s('project','<urn:B>','<urn:hasManager>','<urn:C>','FGAC_ADMIN','OLS_
NET'));
INSERT INTO project_tpl(triple) values
(sdo_rdf_triple_s('project','<urn:A>','<urn:expenseReportAmount>','"100"','FGAC_ADMIN
','OLS_NET'));
INSERT INTO project_tpl(triple) values
(sdo_rdf_triple_s('project','<urn:expenseReportAmount>','rdfs:subPropertyOf','<urn:pr
ojExp>','FGAC_ADMIN','OLS_NET'));
COMMIT;


-- User B uses bulk API to add RDF data
connect b/<password-for-b>
CREATE TABLE  project_stab(RDF$STC_GRAPH varchar2(4000),
RDF$STC_sub varchar2(4000),
RDF$STC_pred varchar2(4000),
RDF$STC_obj varchar2(4000)) compress;
GRANT select on project_stab to fgac_admin;

-- For simplicity, data types are omitted.
INSERT INTO project_stab values(null,
'<urn:B>','<urn:expenseReportAmount>','"200"');
INSERT INTO project_stab values(null, '<urn:proj1>','<urn:deadline>','"2012-12-25"');
EXECUTE
sem_apis.bulk_load_from_staging_table('project','b','project_stab' ,network_owner=>'F
GAC_ADMIN',network_name=>'OLS_NET');

-- As User B, check the contents in the application table
 connect b/<password-for-b>
SELECT * from a.project_tpl order by ctxt1;

SDO_RDF_TRIPLE_S(8.5963E+18, 7, 1.4711E+18, 2.0676E+18, 8.5963E+18)    1000
SDO_RDF_TRIPLE_S(5.1676E+18, 7, 8.5963E+18, 2.0676E+18, 5.1676E+18)    1000
SDO_RDF_TRIPLE_S(2.3688E+18, 7, 1.4711E+18, 4.6588E+18, 2.3688E+18)    1000
SDO_RDF_TRIPLE_S(7.6823E+18, 7, 4.6588E+18, 1.1911E+18, 7.6823E+18)    1000
SDO_RDF_TRIPLE_S(6.6322E+18, 7, 8.5963E+18, 4.6588E+18, 6.6322E+18)    1500
SDO_RDF_TRIPLE_S(8.4800E+18, 7, 6.2294E+18, 5.4118E+18, 8.4800E+18)    1500

6 rows selected.
SELECT count(1) from fgac_admin.ols_net#rdfm_project;
```

```
6

-- As User A, check the contents in the application table
-- As expected, A can only see 4 triples
SQL> conn a/<password>
SQL> select * from a.project_tpl order by ctxt1;
SDO_RDF_TRIPLE_S(8.5963E+18, 7, 1.4711E+18, 2.0676E+18, 8.5963E+18)    1000

SDO_RDF_TRIPLE_S(5.1676E+18, 7, 8.5963E+18, 2.0676E+18, 5.1676E+18)    1000

SDO_RDF_TRIPLE_S(2.3688E+18, 7, 1.4711E+18, 4.6588E+18, 2.3688E+18)    1000

SDO_RDF_TRIPLE_S(7.6823E+18, 7, 4.6588E+18, 1.1911E+18, 7.6823E+18)    1000

SQL> select count(1) fromfgac_admin.ols_net#rdfm_project;
4

-- User C uses incremental APIs to add RDF data including 2 quads
connect c/<password-for-c>
INSERT INTO a.project_tpl(triple) values
(sdo_rdf_triple_s('project','<urn:C>','<urn:expenseReportAmount>','"400"','FGAC_ADMIN
','OLS_NET'));
INSERT INTO a.project_tpl(triple) values
(sdo_rdf_triple_s('project','<urn:proj1>','<urn:hasBudget>','"10000"','FGAC_ADMIN','O
LS_NET'));
INSERT INTO a.project_tpl(triple) values
(sdo_rdf_triple_s('project:<urn:proj2>','<urn:proj2>','<urn:hasBudget>','"20000"','FG
AC_ADMIN','OLS_NET'));
INSERT INTO a.project_tpl(triple) values
(sdo_rdf_triple_s('project:<urn:proj2>','<urn:proj2>','<urn:dependsOn>','<urn:proj1>'
,'FGAC_ADMIN','OLS_NET'));
COMMIT;
```

5. Query the data as different users using the default label.

```
-- Now as user A, B, C, execute the following query
select lpad(nvl(g, ' '), 20) || ' ' || s || ' ' || p || ' ' || o from
table(sem_match('select * where { graph ?g { ?s ?p ?o }}',
sem_models('project'),
null,
null,
null,
null,
'GRAPH_MATCH_UNNAMED=T',
null,
null,
'FGAC_ADMIN',
'OLS_NET'))
    order by g, s, p, o;

connect  a/<password-for-a>
-- Repeat the preceding query
SQL> /

urn:A urn:expenseReportAmount 100
urn:A urn:hasManager urn:B
urn:B urn:hasManager urn:C
urn:expenseReportAmount http://www.w3.org/2000/01/rdf-schema#subPropertyOf
urn:projExp
SQL> connect  b/<password-for-b>
SQL> /

urn:A urn:expenseReportAmount 100
```

```
urn:A urn:hasManager urn:B
urn:B urn:expenseReportAmount 200
urn:B urn:hasManager urn:C
urn:expenseReportAmount http://www.w3.org/2000/01/rdf-schema#subPropertyOf
urn:projExp
urn:proj1 urn:deadline 2012-12-25
SQL> connect  c/<password-for-c>
SQL> /

urn:proj2 urn:proj2 urn:dependsOn urn:proj1
urn:proj2 urn:proj2 urn:hasBudget 20000
urn:A urn:expenseReportAmount 100
urn:A urn:hasManager urn:B
urn:B urn:expenseReportAmount 200
urn:B urn:hasManager urn:C
urn:C urn:expenseReportAmount 400
urn:expenseReportAmount http://www.w3.org/2000/01/rdf-schema#subPropertyOf
urn:projExp
urn:proj1 urn:deadline 2012-12-25
urn:proj1 urn:hasBudget 10000
```

As expected, different users (with different labels) can see different sets of triples in the project RDF graph.

6. Query the same data as user C using different labels.

```
exec sa_utl.set_label('defense',char_to_label('defense','SE'));
exec sa_utl.set_row_label('defense',char_to_label('defense','SE'));
```

The same query used in the preceding step produces just 6 matches with label set to SE:

```
urn:A urn:expenseReportAmount 100
urn:A urn:hasManager urn:B
urn:B urn:expenseReportAmount 200
urn:B urn:hasManager urn:C
urn:expenseReportAmount http://www.w3.org/2000/01/rdf-schema#subPropertyOf
urn:projExp
urn:proj1 urn:deadline 2012-12-25

6 rows selected.
```

If user C picks the weakest label ("unclassified"), then user C sees even less

```
exec sa_utl.set_label('defense',char_to_label('defense','UN'));
exec sa_utl.set_row_label('defense',char_to_label('defense','UN'));
```

The same query used in the preceding step produces just 4 matches:

```
urn:A urn:expenseReportAmount 100
urn:A urn:hasManager urn:B
urn:B urn:hasManager urn:C
urn:expenseReportAmount http://www.w3.org/2000/01/rdf-schema#subPropertyOf
urn:projExp
```

If user C wants to run the query only against triples/quads with data label that dominates "Secret":

```
-- First set the label back
exec sa_utl.set_label('defense',char_to_label('defense','TS'));
exec sa_utl.set_row_label('defense',char_to_label('defense','TS'));

select lpad(nvl(g, ' '), 20) || ' ' || s || ' ' || p || ' ' || o
from table(sem_match('select * where { graph ?g { ?s ?p ?o }}',
sem_models('project'),
```

```
null,
null,
null,
null,
'MIN_LABEL=SE POLICY_NAME=DEFENSE GRAPH_MATCH_UNNAMED=T',
null,
null,
'FGAC_ADMIN',
'OLS_NET'))
order by g, s, p, o;
```

The query response excludes those assertions made by user A:

```
urn:proj2 urn:proj2 urn:dependsOn urn:proj1
urn:proj2 urn:proj2 urn:hasBudget 20000
urn:B urn:expenseReportAmount 200
urn:C urn:expenseReportAmount 400
urn:proj1 urn:deadline 2012-12-25
urn:proj1 urn:hasBudget 10000

6 rows selected.
```

The same query can be executed as User A. However, no matches are returned, as expected.

You can delete RDF data when OLS is enabled for RDF. In the following example, assume that SEM_RDFSA.APPLY_OLS_POLICY has been executed successfully, and that the same user setup and label designs are used as in the preceding example.

```
-- First, create a test RDF graph as user A and grant access to users B and C
connect a/<password-for-a>

create table test_tpl (triple sdo_rdf_triple_s) compress for oltp;
grant select, insert, update, delete on test_tpl to B, C;

-- The following will fail with an error message
-- "Error while creating triggers: If OLS
-- is enabled,  you have to apply table policy
-- before creating an OLS-enabled RDF graph"
--
EXECUTE sem_apis.create_rdf_graph('test', 'test_tpl',
'triple',network_owner=>'FGAC_ADMIN',network_name=>'OLS_NET');

-- Grant select on the RDF graph view to users B and C
grant select on fgac_admin.ols_net#rdfm_test to B,C;

-- You need to run this API first

connect fgac_admin/<password-for-fgac_admin>

EXECUTE sem_ols.apply_policy_to_app_tab('defense', 'A',
'TEST_TPL',network_owner=>'FGAC_ADMIN',network_name=>'OLS_NET');

-- Now RDF graph creation (after OLS policy has been applied) can go through
connect a/<password-for-a>
EXECUTE sem_apis.create_rdf_graph('test', 'test_tpl',
'triple',network_owner=>'FGAC_ADMIN',network_name=>'OLS_NET');

-- Add a triple as User A
INSERT INTO test_tpl(triple) values
(sdo_rdf_triple_s('test','<urn:A>','<urn:p>','<urn:B>','FGAC_ADMIN','OLS_NET'));
COMMIT;
```

```
-- Add the same triple as User B
connect b/<password-for-b>
INSERT INTO a.test_tpl(triple) values
(sdo_rdf_triple_s('test','<urn:A>','<urn:p>','<urn:B>','FGAC_ADMIN','OLS_NET'));
COMMIT;


-- Now User B can see both triples in the application table as well as the RDF graph view
set numwidth 20
SELECT * from a.test_tpl;

SDO_RDF_TRIPLE_S(8596269297967065604, 19, 1471072612573670395, 28121856352072361
78, 8596269297967065604)
                1000

SDO_RDF_TRIPLE_S(8596269297967065604, 19, 1471072612573670395, 28121856352072361
78, 8596269297967065604)
                1500

SELECT count(1) from fgac_admin.ols_net#rdfm_test;
                   2

-- User A can only see one triple due to A's label assignment, as expected.

SELECT * from a.test_tpl;

SDO_RDF_TRIPLE_S(8596269297967065604, 19, 1471072612573670395, 28121856352072361
78, 8596269297967065604)
                1000

SELECT count(1) from fgac_admin.ols_net#rdfm_test;
                   1


-- User A issues a delete to remove A's assertions
SQL> delete from a.test_tpl;
1 row deleted.

COMMIT;
Commit complete.


-- Now user A has no assertions left.

SELECT * from a.test_tpl;
no rows selected

SELECT count(1) from fgac_admin.ols_net#rdfm_test;
                   0

-- Note that the preceding delete does not affect the same assertion made by B.
connect b/<password-for-b>
SELECT * from a.test_tpl;

SDO_RDF_TRIPLE_S(8596269297967065604, 19, 1471072612573670395, 28121856352072361
78, 8596269297967065604)
                1500

SELECT count(1) from fgac_admin.ols_net#rdfm_test;
                   1

-- User B can remove this assertion using a DELETE statement.
```

```
-- The following DELETE statement uses the oracle_orardf_res2vid function
-- to narrow down the scope to triples with a particular subject.
DELETE FROM a.test_tpl app_tab
       where app_tab.triple.rdf_s_id =
             sem_apis.res2vid('FGAC_ADMIN.OLS_NET#RDF_VALUE$','<urn:A>');

1 row deleted.
```

7 RDF Graph Support for Apache Jena

RDF Graph support for Apache Jena (also referred to here as support for Apache Jena) provides a Java-based interface to Oracle Graph RDF Graph by implementing the well-known Jena Graph, RDF graph, and DatasetGraph APIs.

> **Note:**
>
> This feature was previously referred to as the *Jena Adapter for Oracle Database* and the *Jena Adapter*.

Support for Apache Jena extends the RDF data management capabilities of Oracle Database RDF/OWL.

(Apache Jena is an open source framework. For license and copyright conditions, see `http://www.apache.org/licenses/` and `http://www.apache.org/licenses/LICENSE-2.0`.)

The DatasetGraph APIs are for managing named graph data, also referred to as **quads**. In addition, RDF Graph support for Apache Jena provides network analytical functions on top of RDF data through integrating with the Oracle Spatial Network Data Model Graph feature.

This chapter assumes that you are familiar with major concepts explained in RDF Graph Overview and OWL Concepts . It also assumes that you are familiar with the overall capabilities and use of the Jena Java framework. For information about the Jena framework, see `http://jena.apache.org/`, especially the Jena Documentation page. If you use the network analytical function, you should also be familiar with the Network Data Model feature, which is documented in *Oracle Spatial Topology and Network Data Model Developer's Guide*.

> **Note:**
>
> The current RDF Graph support for Apache Jena release has been tested against Apache Jena 3.1.0, and it supports the RDF schema-private networks environment in Release 19c databases. Because of the nature of open source projects, you should not use this support for Apache Jena with later versions of Jena.
>
> Apache Joseki support has been deprecated, although it still is part of the OTN kit distribution for adapter version 3.1.0 with support for Release 19c databases. References to Joseki have been removed from this book for Release 19c, but you can find information about Joseki in previous versions of the book.

- Setting Up the SPARQL Service
  This section explains how to set up a SPARQL web service endpoint by deploying the `fuseki.war` file in WebLogic Server.
- Setting Up the RDF Graph Environment
  To use the support for Apache Jena to perform queries, you can connect as any user (with suitable privileges) and use any RDF graphs in the RDF network.
- SEM_MATCH and RDF Graph Support for Apache Jena Queries Compared
  There are two ways to query RDF data stored in Oracle Database: SEM_MATCH-based SQL statements and SPARQL queries through the support for Apache Jena.
- Retrieving User-Friendly Java Objects from SEM_MATCH or SQL-Based Query Results
  You can query an RDF graph using any of the following approaches.
- Optimized Handling of SPARQL Queries
  This section describes some performance-related features of the support for Apache Jena that can enhance SPARQL query processing. These features are performed automatically by default.
- Additions to the SPARQL Syntax to Support Other Features
  RDF Graph support for Apache Jena allows you to pass in hints and additional query options. It implements these capabilities by overloading the SPARQL namespace prefix syntax by using Oracle-specific namespaces that contain query options.
- Functions Supported in SPARQL Queries through RDF Graph Support for Apache Jena
  SPARQL queries through the support for Apache Jena can use the following kinds of functions.
- SPARQL Update Support
  RDF Graph support for Apache Jena supports SPARQL Update (`http://www.w3.org/TR/sparql11-update/`), also referred to as SPARUL.
- Analytical Functions for RDF Data
  You can perform analytical functions on RDF data by using the `SemNetworkAnalyst` class in the `oracle.spatial.rdf.client.jena` package.
- Support for Server-Side APIs
  This section describes some of the RDF Graph features that are exposed by RDF Graph support for Apache Jena.
- Bulk Loading Using RDF Graph Support for Apache Jena
  To load thousands to hundreds of thousands of RDF/OWL data files into an Oracle database, you can use the `prepareBulk` and `completeBulk` methods in the `OracleBulkUpdateHandler` Java class to simplify the task.
- Automatic Variable Renaming
  Automatic variable renaming can enable certain queries that previously failed to run successfully.
- JavaScript Object Notation (JSON) Format Support
  JavaScript Object Notation (JSON) format is supported for SPARQL query responses. JSON data format is simple, compact, and well suited for JavaScript programs.
- Other Recommendations and Guidelines
  This section contains various recommendations and other information related to SPARQL queries.
- Example Queries Using RDF Graph Support for Apache Jena
  This section includes example queries using the support for Apache Jena. Each example is self-contained: it typically creates a model, creates triples, performs a query that may involve inference, displays the result, and drops the RDF graph.

- SPARQL Gateway and RDF Data
  SPARQL Gateway is a J2EE web application that is included with the support for Apache Jena. It is designed to make RDF data (RDF/OWL/SKOS) easily available to applications that operate on relational and XML data, including Oracle Business Intelligence Enterprise Edition (OBIEE) 11*g*.
- Deploying Fuseki in Apache Tomcat
  To deploy Fuseki in Apache Tomcat, you can use the Tomcat admin web page, or you can just copy the Fuseki `.war` file into the `webapps` folder of Tomcat and it will be automatically deployed.
- ORARDFLDR Utility for Bulk Loading RDF Data
  This section describes using the ORARDFLDR utility program for Bulk Loading RDF Data.

7.1 Setting Up the Software Environment

To use the support for Apache Jena, you must first ensure that the system environment has the necessary software, including Oracle Database with RDF Graph support enabled, Apache Jena 3.12.0, and JDK 1.8 or later.

You can set up the software environment by performing these actions:

1. Install Oracle Database Enterprise Edition with the Oracle Spatial and Partitioning Options.
2. Enable the support for RDF Graph, as explained in Enabling RDF Graph Support.
3. Download RDF Graph support for Apache Jena from Oracle Software Delivery Cloud.
4. Unzip the kit into a temporary directory, such as (on a Linux system) `/tmp/jena_adapter`. (If this temporary directory does not already exist, create it before the unzip operation.)

   The RDF Graph support for Apache Jena has the following top-level directories:

   ```
    |-- examples
    |-- fuseki
    |-- fuseki_web_app
    |-- jar
    |-- javadoc
    |-- joseki
    |-- joseki_web_app
    |-- protege_plugin
    |-- README
    |-- sparqlgateway_web_app
   ```
5. Install JDK 1.8 or later (if not already installed).
6. Ensure that the `JAVA_HOME` environment variable is referencing the JDK installation. For example:

   ```
   setenv JAVA_HOME /usr/local/packages/jdk18/
   ```
7. If the SPARQL service to support the SPARQL protocol is not set up, set it up as explained in Setting Up the SPARQL Service.

After setting up the software environment, ensure that your RDF Graph environment can enable you to use the support for Apache Jena to perform queries, as explained in Setting Up the RDF Graph Environment.

- If You Used a Previous Version of the Support for Apache Jena

7.1.1 If You Used a Previous Version of the Support for Apache Jena

If you used a previous version of the support for Apache Jena, you must drop all functions/procedure installed by previous Jena adapter in user schemas. Installing the new kit will automatically load the updated functions and procedures, which are compatible with new RDF schema private networks in 19c, and with the support in previous releases.

Connect to the user schema that you have used with the previous Jena adapter and execute the following commands to clean the internal functions and procedures. (Some of the functions and procedures referenced in these commands might not exist in the previous installation, so any failed commands can be ignored.)

```
drop procedure ORACLE_ORARDF_S2SGETSRC;
drop procedure ORACLE_ORARDF_S2SGETSRCCLOB;
drop procedure ORACLE_ORARDF_S2SSVR;
drop procedure ORACLE_ORARDF_S2SSVRNG;
drop procedure ORACLE_ORARDF_S2SSVRNGCLOB;
drop procedure ORACLE_ORARDF_GRANT;
drop procedure ORACLE_ORARDF_VID2NAME_TYPE;
drop procedure ORACLE_ORARDF_S2SSVRNGNPV;
drop procedure ORACLE_ORARDF_S2SSVRNGCLOBNPV;
drop function ORACLE_ORARDF_SGC;
drop function ORACLE_ORARDF_SGCCLOB;
drop function ORACLE_ORARDF_S2SUSR;
drop function ORACLE_ORARDF_S2SUSRNG;
drop function ORACLE_ORARDF_S2SUSRNGL;
drop function ORACLE_ORARDF_S2SUSRNGCLOB;
drop function ORACLE_ORARDF_S2SLG;
drop function ORACLE_ORARDF_GETPLIST;
drop function ORACLE_ORARDF_RES2VID;
drop function ORACLE_ORARDF_VID2URI;
```

7.2 Setting Up the SPARQL Service

This section explains how to set up a SPARQL web service endpoint by deploying the `fuseki.war` file in WebLogic Server.

Although there are several ways to deploy applications in WebLogic Server, this topic refers to the autodeploy option.

> **Note:**
>
> If you want to deploy Fuseki in Apache Tomcat instead of WebLogic Server, see Deploying Fuseki in Apache Tomcat.

1. Download and Install Oracle WebLogic Server 12c or later.
2. Ensure that you have Java 8 or later installed.
3. Set the FUSEKI_BASE parameter, which defines the location of the Fuseki configuration files. By default, this parameter is set to `/etc/fuseki`.

You can set this parameter to the the fuseki folder from downloaded OTN kit, which already contains the fuseki configuration files. See the Jena Fuseki documentation for more details: https://jena.apache.org/documentation/fuseki2/fuseki-layout.html

4. Configure an Oracle dataset in the fuseki configuration file: `config.ttl`

   a. Before editing the Fuseki configuration file, create an RDF schema-private network (explained in Schema-Private RDF Networks). For example, assuming a network with name SAMPLE_NET in user schema RDFUSER and tablespace RDFTBS, the following command creates the RDF network.

```
EXECUTE SEM_APIS.CREATE_RDF_NETWORK('RDFTBS',
options=>'MODEL_PARTITIONING=BY_HASH_P MODEL_PARTITIONS=16',
network_owner=>'RDFUSER', network_name=>'SAMPLE_NET' );
```

   b. Edit file config.ttl, and add an oracle:Dataset definition using a model named M_NAMED_GRAPHS. The following snippet shows the configuration. The `oracle:allGraphs` predicate denotes that the SPARQL service endpoint will serve queries using all graphs stored in the M_NAMED_GRAPHS model.

```
<#oracle> rdf:type oracle:Dataset;
oracle:connection
[ a oracle:OracleConnection ;
oracle:jdbcURL "jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)
(HOST=<host>)(PORT=<port>))(CONNECT_DATA=(SERVER=DEDICATED)
(SERVICE_NAME=<service_name>)))";
oracle:User "RDFUSER"
oracle:Password "<password>"
];
oracle:allGraphs [ oracle:firstModel "M_NAMED_GRAPHS";
                   oracle:networkOwner "RDFUSER";
                   oracle:networkName "SAMPLE_NET"] .
```

   c. Link the oracle dataset in the `service` section of the Fuseki configuration file:

```
<#service> rdf:type fuseki:Service ;
    # URI of the dataset -- http://host:port/ds
    fuseki:name                     "oracle" ;

    # SPARQL query services e.g. http://host:port/ds/sparql?query=...
    fuseki:serviceQuery             "sparql" ;
    fuseki:serviceQuery             "query" ;
    # SPARQL Update service -- http://host:port/ds/update?request=...
    fuseki:serviceUpdate            "update" ;   # SPARQL query service
-- /ds/update

    # Upload service -- http://host:port/ds/upload?graph=default or ?
graph=URI or ?default
    # followed by a multipart body, each part being RDF syntax.
    # Syntax determined by the file name extension.
    fuseki:serviceUpload            "upload" ;   # Non-SPARQL upload
service

    # SPARQL Graph store protocol (read and write)
    # GET, PUT, POST DELETE to http://host:port/ds/data?graph= or ?
default=
```

```
        fuseki:serviceReadWriteGraphStore      "data" ;

        # A separate read-only graph store endpoint:
        fuseki:serviceReadGraphStore       "get" ;   # Graph store protocol
    (read only) -- /ds/get

        fuseki:dataset           <#oracle> ;
        .
```

The M_NAMED_GRAPHS model will be created automatically (if it does not already exist) upon the first SPARQL query request. You can add a few example triples and quads to test the named graph functions. For example, for a database before Release 19.3:

```
SQL> CONNECT username/password
SQL> INSERT INTO m_named_graphs_tpl
VALUES(sdo_rdf_triple_s('m_named_graphs','<urn:s>','<urn:p>','<urn:o>'));
SQL> INSERT INTO m_named_graphs_tpl
VALUES(sdo_rdf_triple_s('m_named_graphs:<urn:G1>','<urn:g1_s>','<urn:g1_p>'
,'<urn:g1_o>'));
SQL> INSERT INTO m_named_graphs_tpl
VALUES(sdo_rdf_triple_s('m_named_graphs:<urn:G2>','<urn:g2_s>','<urn:g2_p>'
,'<urn:g2_o>'));
SQL> COMMIT;
```

5. Go to the `autodeploy` directory of WebLogic Server and copy files, as follows. (For information about automatically deploying applications in development domains, see: `http://docs.oracle.com/cd/E24329_01/web.1211/e24443/autodeploy.htm`)

   ```
   cd <domain_name>/autodeploy
   cp -rf  /tmp/jena_adapter/fuseki_web_app/fuseki.war  <domain_name>/autodeploy
   ```

   In the preceding example, <*domain_name*> is the name of a WebLogic Server domain.

   Note that while you can run a WebLogic Server domain in two different modes, development and production, only development mode allows you use the `autodeploy` feature.

6. Verify your deployment by using your Web browser to connect to a URL in the following format (assume that the Web application is deployed at port 7001): `http://<hostname>:7001/fuseki`

   You should see a page titled *Apache Jena Fuseki*, and a list of datasets on the server. This example should show the `/oracle` dataset.

7. Execute the query by clicking on the Query button on the /oracle dataset and entering the following query:

   ```
   SELECT ?g ?s ?p ?o
   WHERE
   { GRAPH ?g { ?s ?p ?o} }
   ```

   The result should be an HTML table with four columns and two sets of result bindings.

- Client Identifiers
- Using OLTP Compression for Application Tables and Staging Tables
- N-Triples Encoding for Non-ASCII Characters

7.2.1 Client Identifiers

For every database connection created or used by the support for Apache Jena, a client identifier is associated with the connection. The client identifier can be helpful, especially in a Real Application Cluster (Oracle RAC) environment, for isolating RDF Graph support for Apache Jena-related activities from other database activities when you are doing performance analysis and tuning.

By default, the client identifier assigned is `JenaAdapter`. However, you can specify a different value by setting the Java VM `clientIdentifier` property using the following format:

```
-Doracle.spatial.rdf.client.jena.clientIdentifier=<identificationString>
```

To start the tracing of only RDF Graph support for Apache Jena-related activities on the database side, you can use the DBMS_MONITOR.CLIENT_ID_TRACE_ENABLE procedure. For example:

```
SQL> EXECUTE DBMS_MONITOR.CLIENT_ID_TRACE_ENABLE('JenaAdapter', true, true);
```

7.2.2 Using OLTP Compression for Application Tables and Staging Tables

By default, the support for Apache Jena creates the application tables and any staging tables (the latter used for bulk loading, as explained in Bulk Loading Using RDF Graph Support for Apache Jena) using basic table compression with the following syntax:

```
CREATE TABLE .... (... column definitions ...) ... compress;
```

However, if you are licensed to use the Oracle Advanced Compression option no the database, you can set the following JVM property to turn on OLTP compression, which compresses data during all DML operations against the underlying application tables and staging tables:

```
-Doracle.spatial.rdf.client.jena.advancedCompression="compress for oltp"
```

7.2.3 N-Triples Encoding for Non-ASCII Characters

For any non-ASCII characters in the lexical representation of RDF resources, `\uHHHH` N-Triples encoding is used when the characters are inserted into the Oracle database. (For details about N-Triples encoding, see `http://www.w3.org/TR/rdf-testcases/#ntrip_grammar`.) Encoding of the constant resources in a SPARQL query is handled in a similar fashion.

Using `\uHHHH` N-Triples encoding enables support for international characters, such as a mix of Norwegian and Swedish characters, in the Oracle database even if a supported Unicode character set is not being used.

7.3 Setting Up the RDF Graph Environment

To use the support for Apache Jena to perform queries, you can connect as any user (with suitable privileges) and use any RDF graphs in the RDF network.

If your RDF Graph environment already meets the requirements, you can go directly to compiling and running Java code that uses the support for Apache Jena. If your RDF Graph environment is not yet set up to be able to use the support for Apache Jena, you can perform actions similar to the following example steps:

1. Connect as SYSTEM:

```
sqlplus system/<password-for-system>
```

2. Create a tablespace for the system tables. For example:

```
CREATE TABLESPACE rdf_users datafile 'rdf_users01.dbf'
    size 128M reuse autoextend on next 64M
    maxsize unlimited segment space management auto;
```

3. Create a database user (for connecting to the database to use the RDF network and the support for Apache Jena). For example:

```
CREATE USER rdfusr IDENTIFIED BY <password-for-udfusr>
                   DEFAULT TABLESPACE rdf_users;
```

4. Grant the necessary privileges to this database user. For example:

```
GRANT connect, resource TO rdfusr;
```

5. Create the RDF network. For example:

```
EXECUTE sem_apis.create_RDF_network('RDF_USERS', network_owner=>'RDFUSR',
network_name=>'LOCALNET');
```

6. To use the support for Apache Jena with your own RDF data, perform the appropriate steps to store data, create an RDF graph, and create database indexes, as explained in Quick Start for Using RDF Data. Then perform queries by compiling and running Java code; see Example Queries Using RDF Graph Support for Apache Jena for information about example queries.

   To use the support for Apache Jena with supplied example data, see Example Queries Using RDF Graph Support for Apache Jena.

7.4 SEM_MATCH and RDF Graph Support for Apache Jena Queries Compared

There are two ways to query RDF data stored in Oracle Database: SEM_MATCH-based SQL statements and SPARQL queries through the support for Apache Jena.

Queries using each approach are similar in appearance, but there are important behavioral differences. To ensure consistent application behavior, you must understand the differences and use care when dealing with query results coming from SEM_MATCH queries and SPARQL queries.

The following simple examples show the two approaches.

Query 1 (SEM_MATCH-based)

```
select s, p, o
    from table(sem_match('{?s ?p ?o}', sem_models('Test_Model'), ....))
```

Query 2 (SPARQL query through Support for Apache Jena)

```
select ?s ?p ?o
where {?s ?p ?o}
```

These two queries perform the same kind of functions; however, there are some important differences. Query 1 (SEM_MATCH-based):

- Reads all triples out of `Test_Model`.
- Does not differentiate among URI, bNode, plain literals, and typed literals, and it does not handle long literals.

- Does not unescape certain characters (such as `'\n'`).

Query 2 (SPARQL query executed through the support for Apache Jena) also reads all triples out of `Test_Model` (assume it executed a call to `ModelOracleSem` referring to the same underlying `Test_Model`). However, Query 2:

- Reads out additional columns (as opposed to just the `s`, `p`, and `o` columns with the SEM_MATCH table function), to differentiate URI, bNodes, plain literals, typed literals, and long literals. This is to ensure proper creation of Jena Node objects.
- Unescapes those characters that are escaped when stored in Oracle Database

Blank node handling is another difference between the two approaches:

- In a SEM_MATCH-based query, blank nodes are always treated as constants.
- In a SPARQL query, a blank node that *is not* wrapped inside < and > is treated as a variable when the query is executed through the support for Apache Jena. This matches the SPARQL standard semantics. However, a blank node that *is* wrapped inside < and > is treated as a constant when the query is executed, and the support for Apache Jena adds a proper prefix to the blank node label as required by the underlying data modeling.

The maximum length for the name of an RDF graph created using the support for Apache Jena API is 22 characters.

7.5 Retrieving User-Friendly Java Objects from SEM_MATCH or SQL-Based Query Results

You can query an RDF graph using any of the following approaches.

- SPARQL (through Java methods or web service end point)
- SEM_MATCH (table function that has SPARQL queries embedded)
- SQL (by querying the *<user>.<network_name>*#RDFM*<model>* view and joining with *<user>.<network_name>*#RDF_VALUE$ and/or other tables)

For Java developers, the results from the first approach are easy to consume. The results from the second and third approaches, however, can be difficult for Java developers because you must parse various columns to get properly typed Java objects that are mapped from typed RDF literals. RDF graph support for Apache Jena supports several methods and helper functions to simplify the task of getting properly typed Java objects from a JDBC result set. These methods and helper functions are shown in the following examples:

- Example 7-1
- Example 7-2
- Example 7-3

These examples use an RDF graph TGRAPH into which a set of typed literals is added through inserts into the RDF graph's RDFT view, as in the following code:

```
exec sem_apis.create_rdf_graph('tgraph',null,null,network_owner=>'RDFUSR',network_name=>'LOCALNET');
exec sem_apis.truncate_rdf_graph('tgraph', network_owner=>'RDFUSR',network_name=>'LOCALNET');

-- Add some triples
insert into LOCALNET#RDFT_TGRAPH(TRIPLE) values(sdo_rdf_triple_s('tgraph','<urn:s1>','<urn:p1>',
'<urn:o1>','RDFUSR','LOCALNET'));
insert into LOCALNET#RDFT_TGRAPH(TRIPLE) values(sdo_rdf_triple_s('tgraph','<urn:s2>','<urn:p2>', '"hello
world"','RDFUSR','LOCALNET'));
insert into LOCALNET#RDFT_TGRAPH(TRIPLE) values(sdo_rdf_triple_s('tgraph','<urn:s3>','<urn:p3>', '"hello
```

```
world"@en','RDFUSR','LOCALNET'));
insert into LOCALNET#RDFT_TGRAPH(TRIPLE) values(sdo_rdf_triple_s('tgraph','<urn:s4>','<urn:p4>', '" o1o
"^^<http://www.w3.org/2001/XMLSchema#string>','RDFUSR','LOCALNET'));
insert into LOCALNET#RDFT_TGRAPH(TRIPLE) values(sdo_rdf_triple_s('tgraph','<urn:s4>','<urn:p4>',
'"xyz"^^<http://mytype>','RDFUSR','LOCALNET'));
insert into LOCALNET#RDFT_TGRAPH(TRIPLE) values(sdo_rdf_triple_s('tgraph','<urn:s5>','<urn:p5>',
'"123"^^<http://www.w3.org/2001/XMLSchema#integer>','RDFUSR','LOCALNET'));
insert into LOCALNET#RDFT_TGRAPH(TRIPLE) values(sdo_rdf_triple_s('tgraph','<urn:s5>','<urn:p5>',
'"123.456"^^<http://www.w3.org/2001/XMLSchema#double>','RDFUSR','LOCALNET'));
insert into LOCALNET#RDFT_TGRAPH(TRIPLE) values(sdo_rdf_triple_s('tgraph','<urn:s6>','<urn:p6>',
'_:bn1','RDFUSR','LOCALNET'));

-- Add some quads
insert into LOCALNET#RDFT_TGRAPH(TRIPLE)
values(sdo_rdf_triple_s('tgraph:<urn:g1>','<urn:s1>','<urn:p1>', '<urn:o1>','RDFUSR','LOCALNET'));
insert into LOCALNET#RDFT_TGRAPH(TRIPLE)
values(sdo_rdf_triple_s('tgraph:<urn:g2>','<urn:s1>','<urn:p1>', '<urn:o1>','RDFUSR','LOCALNET'));
insert into LOCALNET#RDFT_TGRAPH(TRIPLE)
values(sdo_rdf_triple_s('tgraph:<urn:g2>','<urn:s2>','<urn:p2>', '"hello world"','RDFUSR','LOCALNET'));
insert into LOCALNET#RDFT_TGRAPH(TRIPLE)
values(sdo_rdf_triple_s('tgraph:<urn:g2>','<urn:s3>','<urn:p3>', '"hello
world"@en','RDFUSR','LOCALNET'));
insert into LOCALNET#RDFT_TGRAPH(TRIPLE)
values(sdo_rdf_triple_s('tgraph:<urn:g2>','<urn:s4>','<urn:p4>', '" o1o "^^<http://www.w3.org/2001/
XMLSchema#string>','RDFUSR','LOCALNET'));
insert into LOCALNET#RDFT_TGRAPH(TRIPLE)
values(sdo_rdf_triple_s('tgraph:<urn:g2>','<urn:s4>','<urn:p4>', '"xyz"^^<http://
mytype>','RDFUSR','LOCALNET'));
insert into LOCALNET#RDFT_TGRAPH(TRIPLE)
values(sdo_rdf_triple_s('tgraph:<urn:g2>','<urn:s5>','<urn:p5>', '"123"^^<http://www.w3.org/2001/
XMLSchema#integer>','RDFUSR','LOCALNET'));
insert into LOCALNET#RDFT_TGRAPH(TRIPLE)
values(sdo_rdf_triple_s('tgraph:<urn:g2>','<urn:s5>','<urn:p5>', '"123.456"^^<http://www.w3.org/2001/
XMLSchema#double>','RDFUSR','LOCALNET'));
insert into LOCALNET#RDFT_TGRAPH(TRIPLE)
values(sdo_rdf_triple_s('tgraph:<urn:g2>','<urn:s6>','<urn:p6>', '_:bn1','RDFUSR','LOCALNET'));
insert into LOCALNET#RDFT_TGRAPH(TRIPLE)
values(sdo_rdf_triple_s('tgraph:<urn:g2>','<urn:s7>','<urn:p7>', '"2002-10-10T12:00:00-05:00"^^<http://
www.w3.org/2001/XMLSchema#dateTime>','RDFUSR','LOCALNET'));
commit;
```

Example 7-1 SQL-Based Graph Query

Example 7-1 runs a pure SQL-based graph query and constructs Jena objects.

```
iTimeout = 0; // no time out
iDOP = 1;     // degree of parallelism
iStartColPos = 2;

queryString = "select 'hello'||rownum as extra,
o.VALUE_TYPE,o.LITERAL_TYPE,o.LANGUAGE_TYPE,o.LONG_VALUE,o.VALUE_NAME "
            + "  from rdfusr.localnet#rdfm_tgraph g, rdfusr.localnet#rdf_value$ o where
g.canon_end_node_id = o.value_id";

rs = oracle.executeQuery(queryString, iTimeout, iDOP, bindValues);

while (rs.next()) {
  node = OracleSemIterator.retrieveNodeFromRS(rs, iStartColPos,
OracleSemQueryPlan.CONST_FIVE_COL, translator);
  System.out.println("Result " + node.getClass().getName() + " = " + node + " " +
rs.getString(1));
}
```

Example 7-1 might generate the following output:

```
Result org.apache.jena.graph.Node_Literal = "123"^^http://www.w3.org/2001/
XMLSchema#decimal hello1
Result org.apache.jena.graph.Node_Literal = "123"^^http://www.w3.org/2001/
XMLSchema#decimal hello2
Result org.apache.jena.graph.Node_URI = urn:o1 hello3
Result org.apache.jena.graph.Node_URI = urn:o1 hello4
Result org.apache.jena.graph.Node_URI = urn:o1 hello5
Result org.apache.jena.graph.Node_Literal = "hello world" hello6
Result org.apache.jena.graph.Node_Literal = "hello world" hello7
Result org.apache.jena.graph.Node_Literal = "hello world"@en hello8
Result org.apache.jena.graph.Node_Literal = "hello world"@en hello9
Result org.apache.jena.graph.Node_Literal = " o1o " hello10
Result org.apache.jena.graph.Node_Literal = " o1o " hello11
Result org.apache.jena.graph.Node_Literal = "xyz"^^http://mytype hello12
Result org.apache.jena.graph.Node_Literal = "xyz"^^http://mytype hello13
Result org.apache.jena.graph.Node_Literal = "1.23456E2"^^http://www.w3.org/2001/
XMLSchema#double hello14
Result org.apache.jena.graph.Node_Literal = "1.23456E2"^^http://www.w3.org/2001/
XMLSchema#double hello15
Result org.apache.jena.graph.Node_Blank = m15mbn1 hello16
Result org.apache.jena.graph.Node_Blank = m15g3C75726E3A67323Egmbn1 hello17
Result org.apache.jena.graph.Node_Literal = "2002-10-10T17:00:00Z"^^http://www.w3.org/
2001/XMLSchema#dateTime hello18
```

Example 7-2 Hybrid Query Mixing SEM_MATCH with Regular SQL Constructs

Example 7-2 uses the `OracleSemIterator.retrieveNodeFromRS` API to construct a Jena object by reading the five consecutive columns (in the exact order of value type, literal type, language type, long value, and value name), and by performing the necessary unescaping and object instantiations.

```
iStartColPos = 1;
queryString = "select  g$RDFVTYP, g, count(1) as cnt "
            + "  from table(sem_match('{ GRAPH ?g { ?s ?p ?
o . } }',sem_models('tgraph'),null,null,null,null,null,null,null,'RDFUSR','LOCALNET')) "
            + " group by g$RDFVTYP, g";

rs = oracle.executeQuery(queryString, iTimeout, iDOP, bindValues);
while (rs.next()) {
  node = OracleSemIterator.retrieveNodeFromRS(rs, iStartColPos,
OracleSemQueryPlan.CONST_TWO_COL, translator);
  System.out.println("Result " + node.getClass().getName() + " = " + node + " " +
rs.getInt(iStartColPos + 2));
}
```

Example 7-2 might generate the following output:

```
Result org.apache.jena.graph.Node_URI = urn:g2 9
Result org.apache.jena.graph.Node_URI = urn:g1 1
```

In Example 7-2:

- The helper function `executeQuery` in the `Oracle` class is used to run the SQL statement, and the `OracleSemIterator.retrieveNodeFromRS` API (also used in Example 7-1) is used to construct Jena objects.
- Only two columns are used in the output: value type (g$RDFVTYP) and value name (g), it is known that this g variable can never be a literal RDF resource.
- The column order is significant. For a two-column variable, the first column must be the value type and the second column must be the value name.

Example 7-3 SEM_MATCH Query

Example 7-3 runs a SEM_MATCH query and constructs an iterator (instance of `OracleSemIterator`) that returns a list of Jena objects.

```
queryString = "select  g$RDFVTYP, g, s$RDFVTYP, s, p$RDFVTYP, p,
o$RDFVTYP,o$RDFLTYP,o$RDFLANG,o$RDFCLOB,o "
            + "  from table(sem_match('{ GRAPH ?g { ?s ?p ?
o . } }',sem_models('tgraph'),null,null,null,null,null,null,null,'RDFUSR','LOCALNET'))";
guide = new ArrayList<String>();
guide.add(OracleSemQueryPlan.CONST_TWO_COL);
guide.add(OracleSemQueryPlan.CONST_TWO_COL);
guide.add(OracleSemQueryPlan.CONST_TWO_COL);
guide.add(OracleSemQueryPlan.CONST_FIVE_COL);

rs = oracle.executeQuery(queryString, iTimeout, iDOP, bindValues);
osi = new OracleSemIterator(rs);
osi.setGuide(guide);
osi.setTranslator(translator);

while (osi.hasNext()) {
  result = osi.next();
  System.out.println("Result " + result.getClass().getName() + " = " + result);
}
```

Example 7-3 might generate the following output:

```
Result oracle.spatial.rdf.client.jena.Domain = <domain 0:urn:g2 1:urn:s5 2:urn:p5
3:"123"^^http://www.w3.org/2001/XMLSchema#decimal>
Result oracle.spatial.rdf.client.jena.Domain = <domain 0:urn:g2 1:urn:s5 2:urn:p5
3:"1.23456E2"^^http://www.w3.org/2001/XMLSchema#double>
Result oracle.spatial.rdf.client.jena.Domain = <domain 0:urn:g2 1:urn:s7 2:urn:p7
3:"2002-10-10T17:00:00Z"^^http://www.w3.org/2001/XMLSchema#dateTime>
Result oracle.spatial.rdf.client.jena.Domain = <domain 0:urn:g2 1:urn:s2 2:urn:p2
3:"hello world">
Result oracle.spatial.rdf.client.jena.Domain = <domain 0:urn:g2 1:urn:s4 2:urn:p4 3:"
o1o ">
Result oracle.spatial.rdf.client.jena.Domain = <domain 0:urn:g2 1:urn:s4 2:urn:p4
3:"xyz"^^http://mytype>
Result oracle.spatial.rdf.client.jena.Domain = <domain 0:urn:g2 1:urn:s6 2:urn:p6
3:m15g3C75726E3A67323Egmbn1>
Result oracle.spatial.rdf.client.jena.Domain = <domain 0:urn:g2 1:urn:s1 2:urn:p1
3:urn:o1>
Result oracle.spatial.rdf.client.jena.Domain = <domain 0:urn:g1 1:urn:s1 2:urn:p1
3:urn:o1>
Result oracle.spatial.rdf.client.jena.Domain = <domain 0:urn:g2 1:urn:s3 2:urn:p3
3:"hello world"@en>
```

In Example 7-3:

- `OracleSemIterator` takes in a JDBC result set. `OracleSemIterator` needs guidance on parsing all the columns that represent the bind values of SPARQL variables. A guide is simply a list of string values. Two constants have been defined to differentiate a 2-column variable (for subject or predicate position) from a 5-column variable (for object position). A translator is also required.
- Four variables are used in the output. The first three variables are not RDF literal resources, so CONST_TWO_COL is used as their guide. The last variable can be an RDF literal resource, so CONST_FIVE_COL is used as its guide.
- The column order is significant, and it must be as shown in the example.

7.6 Optimized Handling of SPARQL Queries

This section describes some performance-related features of the support for Apache Jena that can enhance SPARQL query processing. These features are performed automatically by default.

It assumes that you are familiar with SPARQL, including the CONSTRUCT feature and property paths.

- Compilation of SPARQL Queries to a Single SEM_MATCH Call
- Optimized Handling of Property Paths

7.6.1 Compilation of SPARQL Queries to a Single SEM_MATCH Call

SPARQL queries involving DISTINCT, OPTIONAL, FILTER, UNION, ORDER BY, and LIMIT are converted to a single Oracle SEM_MATCH table function. If a query cannot be converted directly to SEM_MATCH because it uses SPARQL features not supported by SEM_MATCH (for example, CONSTRUCT), the support for Apache Jena employs a hybrid approach and tries to execute the largest portion of the query using a single SEM_MATCH function while executing the rest using the Jena ARQ query engine.

For example, the following SPARQL query is directly translated to a single SEM_MATCH table function:

```
PREFIX dc:  <http://purl.org/dc/elements/1.1/>
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?person ?name
  WHERE {
                 {?alice foaf:knows ?person . }
               UNION {
                 ?person ?p ?name. OPTIONAL { ?person ?x ?name1 }
                     }
        }
```

However, the following example query is not directly translatable to a single SEM_MATCH table function because of the CONSTRUCT keyword:

```
PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>
CONSTRUCT  { <http://example.org/person#Alice> vcard:FN ?obj }
  WHERE  { { ?x <http://pred/a> ?obj.}
         UNION
         { ?x <http://pred/b> ?obj.}  }
```

In this case, the support for Apache Jena converts the inner UNION query into a single SEM_MATCH table function, and then passes on the result set to the Jena ARQ query engine for further evaluation.

7.6.2 Optimized Handling of Property Paths

As defined in Jena, a property path is a possible route through an RDF graph between two graph nodes. Property paths are an extension of SPARQL and are more expressive than basic graph pattern queries, because regular expressions can be used over properties for pattern matching RDF graphs. For more information about property paths, see the documentation for the Jena ARQ query engine.

RDF graph support for Apache Jena supports all Jena property path types through the integration with the Jena ARQ query engine, but it converts some common path types directly to native SQL hierarchical queries (not based on SEM_MATCH) to improve performance. The following types of property paths are directly converted to SQL by the support for Apache Jena when dealing with triple data:

- Predicate alternatives: (p1 | p2 | … | p*n*) where p*i* is a property URI
- Predicate sequences: (p1 / p2 / … / p*n*) where p*i* is a property URI
- Reverse paths : (^ p) where p is a predicate URI
- Complex paths: p+, p*, p{0, *n*} where p could be an alternative, sequence, reverse path, or property URI

Path expressions that cannot be captured in this grammar are not translated directly to SQL by the support for Apache Jena, and they are answered using the Jena query engine.

The following example contains a code snippet using a property path expression with path sequences:

```
String m = "PROP_PATH";

ModelOracleSem model = ModelOracleSem.createOracleSemModel(oracle, m);

GraphOracleSem graph = new GraphOracleSem(oracle, m);

// populate the RDF Graph
    graph.add(Triple.create(Node.createURI("http://a"),
    Node.createURI("http://p1"),
    Node.createURI("http://b")));

graph.add(Triple.create(Node.createURI("http://b"),
 Node.createURI("http://p2"),
 Node.createURI("http://c")));

graph.add(Triple.create(Node.createURI("http://c"),
 Node.createURI("http://p5"),
 Node.createURI("http://d")));

String query =
" SELECT ?s  " +
" WHERE {?s (<http://p1>/<http://p2>/<http://p5>)+ <http://d>.}";

QueryExecution qexec =
      QueryExecutionFactory.create(QueryFactory.create(query,
 Syntax.syntaxARQ), model);

try {
  ResultSet results = qexec.execSelect();
  ResultSetFormatter.out(System.out, results);
}
finally {
  if (qexec != null)
    qexec.close();
}

OracleUtils.dropSemanticModel(oracle, m);
model.close();
```

7.7 Additions to the SPARQL Syntax to Support Other Features

RDF Graph support for Apache Jena allows you to pass in hints and additional query options. It implements these capabilities by overloading the SPARQL namespace prefix syntax by using Oracle-specific namespaces that contain query options.

The namespaces are in the form `PREFIX ORACLE_SEM_`xx`_NS`, where *xx* indicates the type of feature (such as `HT` for hint or `AP` for additional predicate)

- SQL Hints
- Using Bind Variables in SPARQL Queries
- Additional WHERE Clause Predicates
- Additional Query Options
- Midtier Resource Caching

7.7.1 SQL Hints

SQL hints can be passed to a SEM_MATCH query including a line in the following form:

```
PREFIX ORACLE_SEM_HT_NS: <http://oracle.com/semtech#hint>
```

Where *hint* can be any hint supported by SEM_MATCH. For example:

```
PREFIX ORACLE_SEM_HT_NS: <http://oracle.com/semtech#leading(t0,t1)>
SELECT ?book ?title ?isbn
WHERE { ?book <http://title> ?title. ?book <http://ISBN> ?isbn }
```

In this example, `t0,t1` refers to the first and second patterns in the query.

Note the slight difference in specifying hints when compared to SEM_MATCH. Due to restrictions of namespace value syntax, a comma (`,`) must be used to separate `t0` and `t1` (or other hint components) instead of a space.

For more information about using SQL hints, see Using the SEM_MATCH Table Function to Query RDF Data, specifically the material about the `HINT0` keyword in the `options` attribute.

7.7.2 Using Bind Variables in SPARQL Queries

In Oracle Database, using bind variables can reduce query parsing time and increase query efficiency and concurrency. Bind variable support in SPARQL queries is provided through namespace pragma specifications similar to ORACLE_SEM_FS_NS.

Consider a case where an application runs two SPARQL queries, where the second (Query 2) depends on the partial or complete results of the first (Query 1). Some approaches that do not involve bind variables include:

- Iterating through results of Query 1 and generating a set of queries. (However, this approach requires as many queries as the number of results of Query 1.)
- Constructing a SPARQL filter expression based on results of Query 1.
- Treating Query 1 as a subquery.

Another approach in this case is to use bind variables, as in the following sample scenario:

Query 1:

```
  SELECT ?x
    WHERE { ... <some complex query> ... };
```

Query 2:

```
  SELECT ?subject ?x
    WHERE {?subject <urn:related> ?x .};
```

The following example shows Query 2 with the syntax for using bind variables with the support for Apache Jena:

```
PREFIX ORACLE_SEM_FS_NS: <http://oracle.com/semtech#no_fall_back,s2s>
PREFIX ORACLE_SEM_UEAP_NS: <http://oracle.com/semtech#x$RDFVID%20in(?,?,?)>
PREFIX ORACLE_SEM_UEPJ_NS: <http://oracle.com/semtech#x$RDFVID>
PREFIX ORACLE_SEM_UEBV_NS: <http://oracle.com/semtech#1,2,3>
SELECT ?subject ?x
WHERE {
  ?subject <urn:related>  ?x
};
```

This syntax includes using the following namespaces:

- ORACLE_SEM_UEAP_NS is like ORACLE_SEM_AP_NS, but the value portion of ORACLE_SEM_UEAP_NS is URL Encoded. Before the value portion is used, it must be URL decoded, and then it will be treated as an additional predicate to the SPARQL query.

  In this example, after URL decoding, the value portion (following the `#` character) of this ORACLE_SEM_UEAP_NS prefix becomes "`x$RDFVID in(?,?,?)`". The three question marks imply a binding to three values coming from Query 1.
- ORACLE_SEM_UEPJ_NS specifies the additional projections involved. In this case, because ORACLE_SEM_UEAP_NS references the x$RDFVID column, which does not appear in the SELECT clause of the query, it must be specified. Multiple projections are separated by commas.
- ORACLE_SEM_UEBV_NS specifies the list of bind values that are URL encoded first, and then concatenated and delimited by commas.

Conceptually, the preceding example query is equivalent to the following non-SPARQL syntax query, in which 1, 2, and 3 are treated as bind values:

```
SELECT ?subject ?x
  WHERE {
    ?subject <urn:related>  ?x
  }
  AND ?x$RDFVID in (1,2,3);
```

In the preceding SPARQL example of Query 2, the three integers 1, 2, and 3 come from Query 1. You can use the `oext:build-uri-for-id` function to generate such internal integer IDs for RDF resources. The following example gets the internal integer IDs from Query 1:

```
PREFIX oext: <http://oracle.com/semtech/jena-adaptor/ext/function#>
SELECT ?x  (oext:build-uri-for-id(?x) as ?xid)
WHERE { ... <some complex query> ... };
```

The values of `?xid` have the form of <rdfvid:*integer-value*>. The application can strip out the angle brackets and the "rdfvid:" strings to get the integer values and pass them to Query 2.

Consider another case, with a single query structure but potentially many different constants. For example, the following SPARQL query finds the hobby for each user who has a hobby and who logs in to an application. Obviously, different users will provide different <uri> values to this SPARQL query, because users of the application are represented using different URIs.

```
SELECT ?hobby
  WHERE { <uri> <urn:hasHobby> ?hobby };
```

One approach, which would not use bind variables, is to generate a different SPARQL query for each different <uri> value. For example, user Jane Doe might trigger the execution of the following SPARQL query:

```
SELECT ?hobby WHERE {
<http://www.example.com/Jane_Doe> <urn:hasHobby> ?hobby };
```

However, another approach is to use bind variables, as in the following example specifying user Jane Doe:

```
PREFIX ORACLE_SEM_FS_NS: <http://oracle.com/semtech#no_fall_back,s2s>
PREFIX ORACLE_SEM_UEAP_NS: <http://oracle.com/
semtech#subject$RDFVID%20in(ORACLE_ORARDF_RES2VID(?))>
PREFIX ORACLE_SEM_UEPJ_NS: <http://oracle.com/semtech#subject$RDFVID>
PREFIX ORACLE_SEM_UEBV_NS: <http://oracle.com/
semtech#http%3a%2f%2fwww.example.com%2fJohn_Doe>
SELECT ?subject ?hobby
  WHERE {
    ?subject <urn:hasHobby>  ?hobby
  };
```

Conceptually, the preceding example query is equivalent to the following non-SPARQL syntax query, in which `http://www.example.com/Jane_Doe` is treated as a bind variable:

```
SELECT ?subject ?hobby
WHERE {
  ?subject <urn:hasHobby>  ?hobby
}
AND ?subject$RDFVID in (ORACLE_ORARDF_RES2VID('http://www.example.com/Jane_Doe'));
```

In this example, ORACLE_ORARDF_RES2VID is a function that translates URIs and literals into their internal integer ID representation. This function is created automatically when the support for Apache Jena is used to connect to an Oracle database.

7.7.3 Additional WHERE Clause Predicates

The SEM_MATCH `filter` attribute can specify additional selection criteria as a string in the form of a WHERE clause without the WHERE keyword. Additional WHERE clause predicates can be passed to a SEM_MATCH query including a line in the following form:

```
PREFIX ORACLE_SEM_AP_NS: <http://oracle.com/semtech#pred>
```

Where *pred* reflects the WHERE clause content to be appended to the query. For example:

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ORACLE_SEM_AP_NS:<http://www.oracle.com/semtech#label$RDFLANG='fr'>
SELECT DISTINCT ?inst ?label
  WHERE { ?inst a <http://someCLass>. ?inst rdfs:label ?label . }
  ORDER BY (?label) LIMIT 20
```

In this example, a restriction is added to the query that the language type of the label variable must be `'fr'`.

7.7.4 Additional Query Options

Additional query options can be passed to a SEM_MATCH query including a line in the following form:

```
PREFIX ORACLE_SEM_FS_NS: <http://oracle.com/semtech#option>
```

Where *option* reflects a query option (or multiple query options delimited by commas) to be appended to the query. For example:

```
PREFIX ORACLE_SEM_FS_NS:
<http://oracle.com/semtech#timeout=3,dop=4,INF_ONLY,ORDERED,ALLOW_DUP=T>
SELECT * WHERE {?subject ?property ?object }
```

The following query options are supported:

- `ALLOW_DUP=t` chooses a faster way to query multiple RDF graphs, although duplicate results may occur.
- `BEST_EFFORT_QUERY=t`, when used with the `TIMEOUT=n` option, returns all matches found in *n* seconds for the SPARQL query.
- `DEGREE=`*n* specifies, at the statement level, the degree of parallelism (*n*) for the query. With multi-core or multi-CPU processors, experimenting with different `DOP` values (such as 4 or 8) may improve performance.

  Contrast `DEGREE` with `DOP`, which specifies parallelism at the session level. `DEGREE` is recommended over `DOP` for use with the support for Apache Jena, because `DEGREE` involves less processing overhead.
- `DOP=`*n* specifies, at the session level, the degree of parallelism (*n*) for the query. With multi-core or multi-CPU processors, experimenting with different `DOP` values (such as 4 or 8) may improve performance.
- `FETCH_SIZE=`*n* specifies the JDBC fetch size parameter (the number of rows to be read from the result set and put in memory on one trip to the database). This parameter can be used to improve performance. A higher value means fewer trips to the database to retrieve all results. The default value is 1000.
- `INF_ONLY` causes only the inferred model to be queried.
- `JENA_EXECUTOR` disables the compilation of SPARQL queries to SEM_MATCH (or native SQL); instead, the Jena native query executor will be used.
- `JOIN=n` specifies how results from a SPARQL SERVICE call to a federated query can be joined with other parts of the query. For information about federated queries and the `JOIN` option, see JOIN Option and Federated Queries.
- `NO_FALL_BACK` causes the underlying query execution engine not to fall back on the Jena execution mechanism if a SQL exception occurs.
- `ODS=n` specifies, at the statement level, the level of dynamic sampling. (For an explanation of dynamic sampling, see the section about estimating statistics with dynamic sampling in *Oracle Database SQL Tuning Guide*.) Valid values for *n* are 1 through 10. For example, you could try `ODS=3` for complex queries.
- `ORDERED` is translated to a LEADING SQL hint for the query triple pattern joins, while performing the necessary RDF_VALUE$ joins last.
- `PLAIN_SQL_OPT=F` disables the native compilation of queries directly to SQL.

- `QID=n` specifies a query ID number; this feature can be used to cancel the query if it is not responding.
- `RESULT_CACHE` uses the Oracle RESULT_CACHE directive for the query.
- `REWRITE=F` disables ODCI_Table_Rewrite for the SEM_MATCH table function.
- `S2S` (SPARQL to pure SQL) causes the underlying SEM_MATCH-based query or queries generated based on the SPARQL query to be further converted into SQL queries ***without*** using the SEM_MATCH table function. The resulting SQL queries are executed by the Oracle cost-based optimizer, and the results are processed by the support for Apache Jena before being passed on to the client. For more information about the `S2S` option, including benefits and usage information, see S2S Option Benefits and Usage Information.

  `S2S` is enabled by default for all SPARQL queries. If you want to disable `S2S`, set the following JVM system property:

  ```
  -Doracle.spatial.rdf.client.jena.defaultS2S=false
  ```
- `SKIP_CLOB=T` causes CLOB values not to be returned for the query.
- `STRICT_DEFAULT=F` allows the default graph to include triples in named graphs. (By default, `STRICT_DEFAULT=T` restricts the default graph to unnamed triples when no data set information is specified.)
- `TIMEOUT=n` (query timeout) specifies the number of seconds (*n*) that the query will run until it is terminated. The underlying SQL generated from a SPARQL query can return many matches and can use features like subqueries and assignments, all of which can take considerable time. The `TIMEOUT` and `BEST_EFFORT_QUERY=t` options can be used to prevent what you consider excessive processing time for the query.
- JOIN Option and Federated Queries
- S2S Option Benefits and Usage Information

7.7.4.1 JOIN Option and Federated Queries

A SPARQL federated query, as described in W3C documents, is a query "over distributed data" that entails "querying one source and using the acquired information to constrain queries of the next source." For more information, see *SPARQL 1.1 Federation Extensions* (`http://www.w3.org/2009/sparql/docs/fed/service`).

You can use the JOIN option (described in Additional Query Options) and the SERVICE keyword in a federated query that uses the support for Apache Jena. For example, assume the following query:

```
SELECT ?s ?s1 ?o
 WHERE { ?s1 ?p1 ?s .
                    {
                     SERVICE <http://sparql.org/books> { ?s ?p ?o }
                    }
                 }
```

If the *local* query portion (`?s1 ?p1 ?s,`) is very selective, you can specify `join=2`, as shown in the following query:

```
PREFIX ORACLE_SEM_FS_NS:   <http://oracle.com/semtech#join=2>
SELECT ?s ?s1 ?o
 WHERE { ?s1 ?p1 ?s .
                    {
                     SERVICE <http://sparql.org/books> { ?s ?p ?o }
```

```
              }
            }
```

In this case, the local query portion (?s1 ?p1 ?s,) is executed locally against the Oracle database. Each binding of ?s from the results is then pushed into the SERVICE part (remote query portion), and a call is made to the service endpoint specified. Conceptually, this approach is somewhat like nested loop join.

If the *remote* query portion (?s ?s1 ?o) is very selective, you can specify join=3, as shown in the following query, so that the remote portion is executed first and results are used to drive the execution of local portion:

```
PREFIX ORACLE_SEM_FS_NS:   <http://oracle.com/semtech#join=3>
SELECT ?s ?s1 ?o
 WHERE { ?s1 ?p1 ?s .
                    {
                     SERVICE <http://sparql.org/books> { ?s ?p ?o }
                    }
                  }
```

In this case, a single call is made to the remote service endpoint and each binding of ?s triggers a local query. As with join=2, this approach is conceptually a nested loop based join, but the difference is that the order is switched.

If neither the local query portion nor the remote query portion is very selective, then we can choose join=1, as shown in the following query:

```
PREFIX ORACLE_SEM_FS_NS:   <http://oracle.com/semtech#join=1>
SELECT ?s ?s1 ?o
 WHERE { ?s1 ?p1 ?s .
                    {
                     SERVICE <http://sparql.org/books> { ?s ?p ?o }
                    }
                }
```

In this case, the remote query portion and the local portion are executed independently, and the results are joined together by Jena. Conceptually, this approach is somewhat like a hash join.

For debugging or tracing federated queries, you can use the HTTP Analyzer in Oracle JDeveloper to see the underlying SERVICE calls.

7.7.4.2 S2S Option Benefits and Usage Information

The S2S option, described in Additional Query Options, provides the following potential benefits:

- It works well with the RESULT_CACHE option to improve query performance. Using the S2S and RESULT_CACHE options is especially helpful for queries that are executed frequently.
- It reduces the parsing time of the SEM_MATCH table function, which can be helpful for applications that involve many dynamically generated SPARQL queries.
- It eliminates the limit of 4000 bytes for the query body (the first parameter of the SEM_MATCH table function), which means that longer, more complex queries are supported.

The S2S option causes an internal in-memory cache to be used for translated SQL query statements. The default size of this internal cache is 1024 (that is, 1024 SQL queries); however, you can adjust the size by using the following Java VM property:

```
-Doracle.spatial.rdf.client.jena.queryCacheSize=<size>
```

7.7.5 Midtier Resource Caching

When RDF data is stored, all of the resource values are hashed into IDs, which are stored in the triples table. The mappings from value IDs to full resource values are stored in the RDF_VALUE$ table. At query time, for each selected variable, Oracle Database must perform a join with the RDF_VALUE$ table to retrieve the resource.

However, to reduce the number of joins, you can use the midtier cache option, which causes an in-memory cache on the middle tier to be used for storing mappings between value IDs and resource values. To use this feature, include the following PREFIX pragma in the SPARQL query:

```
PREFIX ORACLE_SEM_FS_NS: <http://oracle.com/semtech#midtier_cache>
```

To control the maximum size (in bytes) of the in-memory cache, use the `oracle.spatial.rdf.client.jena.cacheMaxSize` system property. The default cache maximum size is 1GB.

Midtier resource caching is most effective for queries using ORDER BY or DISTINCT (or both) constructs, or queries with multiple projection variables. Midtier cache can be combined with the other options specified in Additional Query Options.

If you want to pre-populate the cache with all of the resources in an RDF graph, use the `GraphOracleSem.populateCache` or `DatasetGraphOracleSem.populateCache` method. Both methods take a parameter specifying the number of threads used to build the internal midtier cache. Running either method in parallel can significantly increase the cache building performance on a machine with multiple CPUs (cores).

7.8 Functions Supported in SPARQL Queries through RDF Graph Support for Apache Jena

SPARQL queries through the support for Apache Jena can use the following kinds of functions.

- Functions in the function library of the Jena ARQ query engine
- Native Oracle Database functions for projected variables
- User-defined functions

- Functions in the ARQ Function Library
- Native Oracle Database Functions for Projected Variables
- User-Defined Functions

7.8.1 Functions in the ARQ Function Library

SPARQL queries through the support for Apache Jena can use functions in the function library of the Jena ARQ query engine. These queries are executed in the middle tier.

The following examples use the `upper-case` and `namespace` functions. In these examples, the prefix `fn` is `<http://www.w3.org/2005/xpath-functions#>` and the prefix `afn` is `<http://jena.hpl.hp.com/ARQ/function#>`.

```
PREFIX  fn: <http://www.w3.org/2005/xpath-functions#>
PREFIX  afn: <http://jena.hpl.hp.com/ARQ/function#>
```

```
SELECT (fn:upper-case(?object) as ?object1)
WHERE { ?subject dc:title ?object }

PREFIX  fn: <http://www.w3.org/2005/xpath-functions#>
PREFIX  afn: <http://jena.hpl.hp.com/ARQ/function#>
SELECT ?subject (afn:namespace(?object) as ?object1)
WHERE { ?subject <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> ?object }
```

7.8.2 Native Oracle Database Functions for Projected Variables

SPARQL queries through the support for Apache Jena can use native Oracle Database functions for projected variables. These queries and the functions are executed inside the database. Note that the functions described in this section should not be used together with ARQ functions (described in Functions in the ARQ Function Library).

This section lists the supported native functions and provides some examples. In the examples, the prefix `oext` is `<http://oracle.com/semtech/jena-adaptor/ext/function#>`.

> **Note:**
>
> In the preceding URL, note the spelling `jena-adaptor`, which is retained for compatibility with existing applications and which must be used in queries. The *adapter* spelling is used in regular text, to follow Oracle documentation style guidelines.

- **`oext:upper-literal`** converts literal values (except for long literals) to uppercase. For example:

  ```
  PREFIX  oext: <http://oracle.com/semtech/jena-adaptor/ext/function#>
  SELECT (oext:upper-literal(?object) as ?object1)
  WHERE { ?subject dc:title ?object }
  ```

- **`oext:lower-literal`** converts literal values (except for long literals) to lowercase. For example:

  ```
  PREFIX  oext: <http://oracle.com/semtech/jena-adaptor/ext/function#>
  SELECT (oext:lower-literal(?object) as ?object1)
  WHERE { ?subject dc:title ?object }
  ```

- **`oext:build-uri-for-id`** converts the value ID of a URI, bNode, or literal into a URI form. For example:

  ```
  PREFIX  oext: <http://oracle.com/semtech/jena-adaptor/ext/function#>
  SELECT (oext:build-uri-for-id(?object) as ?object1)
  WHERE { ?subject dc:title ?object }
  ```

  An example of the output might be: `<rdfvid:1716368199350136353>`

  One use of this function is to allow Java applications to maintain in memory a mapping of those value IDs to the lexical form of URIs, bNodes, or literals. The RDF_VALUE$ table provides such a mapping in Oracle Database.

  For a given variable `?var`, if only `oext:build-uri-for-id(?var)` is projected, the query performance is likely to be faster because fewer internal table join operations are needed to answer the query.

- **`oext:literal-strlen`** returns the length of literal values (except for long literals). For example:

```
PREFIX  oext: <http://oracle.com/semtech/jena-adaptor/ext/function#>
SELECT (oext:literal-strlen(?object) as ?objlen)
WHERE { ?subject dc:title ?object }
```

7.8.3 User-Defined Functions

SPARQL queries through the support for Apache Jena can use user-defined functions that are stored in the database.

In the following example, assume that you want to define a string length function (`my_strlen`) that handles long literals (CLOB) as well as short literals. On the SPARQL query side, this function can be referenced under the namespace of `ouext`, which is `http://oracle.com/semtech/jena-adaptor/ext/user-def-function#`.

```
PREFIX  ouext: <http://oracle.com/semtech/jena-adaptor/ext/user-def-function#>
SELECT ?subject ?object (ouext:my_strlen(?object) as ?obj1)
WHERE { ?subject dc:title ?object }
```

Inside the database, functions including `my_strlen`, `my_strlen_cl`, `my_strlen_la`, `my_strlen_lt`, and `my_strlen_vt` are defined to implement this capability. Conceptually, the return values of these functions are mapped as shown in Table 7-1.

Table 7-1 Functions and Return Values for my_strlen Example

Function Name	Return Value
my_strlen	<VAR>
my_strlen_cl	<VAR>$RDFCLOB
my_strlen_la	<VAR>$RDFLANG
my_strlen_lt	<VAR>$RDFLTYP
my_strlen_vt	<VAR>$RDFVTYP

A set of functions (five in all) is used to implement a user-defined function that can be referenced from SPARQL, because this aligns with the internal representation of an RDF resource (in RDF_VALUE$). There are five major columns describing an RDF resource in terms of its value, language, literal type, long value, and value type, and these five columns can be selected out using SEM_MATCH. In this context, a user-defined function simply converts one RDF resource that is represented by five columns to another RDF resource.

These functions are defined as follows:

```
create or replace function my_strlen(rdfvtyp in varchar2,
                              rdfltyp in varchar2,
                              rdflang in varchar2,
                              rdfclob in clob,
                              value   in varchar2
                              ) return varchar2
 as
   ret_val  varchar2(4000);
 begin
   -- value
   if (rdfvtyp = 'LIT') then
     if (rdfclob is null) then
       return length(value);
     else
       return dbms_lob.getlength(rdfclob);
     end if;
   else
```

```
    -- Assign -1 for non-literal values so that application can
    -- easily differentiate
    return '-1';
  end if;
end;
/

create or replace function my_strlen_cl(rdfvtyp in varchar2,
                             rdfltyp in varchar2,
                             rdflang in varchar2,
                             rdfclob in clob,
                             value   in varchar2
                             ) return clob
as
begin
  return null;
end;
/

create or replace function my_strlen_la(rdfvtyp in varchar2,
                             rdfltyp in varchar2,
                             rdflang in varchar2,
                             rdfclob in clob,
                             value   in varchar2
                             ) return varchar2
as
begin
  return null;
end;
/

create or replace function my_strlen_lt(rdfvtyp in varchar2,
                             rdfltyp in varchar2,
                             rdflang in varchar2,
                             rdfclob in clob,
                             value   in varchar2
                             ) return varchar2
as
  ret_val  varchar2(4000);
begin
  -- literal type
  return 'http://www.w3.org/2001/XMLSchema#integer';
end;
/

create or replace function my_strlen_vt(rdfvtyp in varchar2,
                             rdfltyp in varchar2,
                             rdflang in varchar2,
                             rdfclob in clob,
                             value   in varchar2
                             ) return varchar2
as
  ret_val  varchar2(3);
begin
  return 'LIT';
end;
/
```

User-defined functions can also accept a parameter of VARCHAR2 type. The following five functions together define a `my_shorten_str` function that accepts an integer (in VARCHAR2 form) for the substring length and returns the substring. (The substring in this example is 12 characters, and it must not be greater than 4000 bytes.)

```
-- SPARQL query that returns the first 12 characters of literal values.
--
PREFIX  ouext: <http://oracle.com/semtech/jena-adaptor/ext/user-def-function#>
SELECT (ouext:my_shorten_str(?object, "12") as ?obj1) ?subject
WHERE { ?subject dc:title ?object }

create or replace function my_shorten_str(rdfvtyp in varchar2,
                            rdfltyp in varchar2,
                            rdflang in varchar2,
                            rdfclob in clob,
                            value   in varchar2,
                            arg     in varchar2
                            ) return varchar2
as
 ret_val  varchar2(4000);
begin
 -- value
 if (rdfvtyp = 'LIT') then
   if (rdfclob is null) then
     return substr(value, 1, to_number(arg));
   else
     return dbms_lob.substr(rdfclob, to_number(arg), 1);
   end if;
 else
   return null;
 end if;
end;
/

create or replace function my_shorten_str_cl(rdfvtyp in varchar2,
                            rdfltyp in varchar2,
                            rdflang in varchar2,
                            rdfclob in clob,
                            value   in varchar2,
                            arg     in varchar2
                            ) return clob
as
 ret_val  clob;
begin
 -- lob
 return null;
end;
/

create or replace function my_shorten_str_la(rdfvtyp in varchar2,
                            rdfltyp in varchar2,
                            rdflang in varchar2,
                            rdfclob in clob,
                            value   in varchar2,
                            arg     in varchar2
                            ) return varchar2
as
 ret_val  varchar2(4000);
begin
 -- lang
 if (rdfvtyp = 'LIT') then
   return rdflang;
 else
   return null;
 end if;
end;
/
```

```
create or replace function my_shorten_str_lt(rdfvtyp in varchar2,
                            rdfltyp in varchar2,
                            rdflang in varchar2,
                            rdfclob in clob,
                            value   in varchar2,
                            arg     in varchar2
                            ) return varchar2
as
 ret_val  varchar2(4000);
begin
 -- literal type
 ret_val := rdfltyp;
 return ret_val;
end;
/

create or replace function my_shorten_str_vt(rdfvtyp in varchar2,
                            rdfltyp in varchar2,
                            rdflang in varchar2,
                            rdfclob in clob,
                            value   in varchar2,
                            arg     in varchar2
                            ) return varchar2
as
 ret_val  varchar2(3);
begin
 return 'LIT';
end;
/
```

7.9 SPARQL Update Support

RDF Graph support for Apache Jena supports SPARQL Update (`http://www.w3.org/TR/sparql11-update/`), also referred to as SPARUL.

The primary programming APIs involve the Jena class `org.apache.jena.update.UpdateAction` and RDF Graph support for Apache Jena classes `GraphOracleSem` and `DatasetGraphOracleSem`. Example 7-4 shows a SPARQL Update operation removes all triples in named graph `<http://example/graph>` from the relevant RDF graph stored in the database.

Example 7-4 Simple SPARQL Update

```
GraphOracleSem graphOracleSem = .... ;
DatasetGraphOracleSem dsgos = DatasetGraphOracleSem.createFrom(graphOracleSem);

// SPARQL Update operation
String szUpdateAction = "DROP GRAPH <http://example/graph>";

// Execute the Update against a DatasetGraph instance (can be a Jena Model as well)
UpdateAction.parseExecute(szUpdateAction, dsgos);
```

Note that Oracle Database does not keep any information about an empty named graph. This implies if you invoke CREATE GRAPH <*graph_name*> without adding any triples into this graph, then no additional rows in the application table or the underlying RDF_LINK$ table will be created. To an Oracle database, you can safely skip the CREATE GRAPH step, as is the case in Example 7-4.

Example 7-5 SPARQL Update with Insert and Delete Operations

Example 7-5 shows a SPARQL Update operation (from ARQ 2.8.8) involving multiple insert and delete operations.

```
PREFIX : <http://example/>
CREATE GRAPH <http://example/graph> ;
INSERT DATA { :r :p 123 } ;
INSERT DATA { :r :p 1066 } ;
DELETE DATA { :r :p 1066 } ;
INSERT DATA {
  GRAPH <http://example/graph> { :r :p 123 . :r :p 1066 }
} ;
DELETE DATA {
  GRAPH <http://example/graph>  { :r :p 123 }
}
```

After running the update operation in Example 7-5 against an empty `DatasetGraphOracleSem`, running the SPARQL query `SELECT ?s ?p ?o WHERE {?s ?p ?o}` generates the following response:

```
-----------------------------------------------------------------------------------------
------
| s                    | p                    |
o                                                      |
=========================================================================================
======
| <http://example/r> | <http://example/p> | "123"^^<http://www.w3.org/2001/
XMLSchema#decimal> |
-----------------------------------------------------------------------------------------
------
```

Using the same data, running the SPARQL query `SELECT ?g ?s ?p ?o where {GRAPH ?g {?s ?p ?o}}` generates the following response:

```
-----------------------------------------------------------------------------------------
--------------------------------
| g                      | s                    | p                    |
o                                                   |
=========================================================================================
================================
| <http://example/graph> | <http://example/r> | <http://example/p> | "1066"^^<http://
www.w3.org/2001/XMLSchema#decimal> |
-----------------------------------------------------------------------------------------
--------------------------------
```

7.10 Analytical Functions for RDF Data

You can perform analytical functions on RDF data by using the `SemNetworkAnalyst` class in the `oracle.spatial.rdf.client.jena` package.

This support integrates the Network Data Model Graph logic with the underlying RDF data structures. Therefore, to use analytical functions on RDF data, you must be familiar with the Network Data Model Graph feature, which is documented in *Oracle Spatial Topology and Network Data Model Developer's Guide*.

The required NDM Java libraries, including `sdonm.jar` and `sdoutl.jar`, are under the directory `$ORACLE_HOME/md/jlib`. Note that `xmlparserv2.jar` (under `$ORACLE_HOME/xdk/lib`) must be included in the `classpath` definition.

Example 7-6 Performing Analytical functions on RDF Data

Example 7-6 uses the `SemNetworkAnalyst` class, which internally uses the NDM `NetworkAnalyst` API

```
Oracle oracle = new Oracle(jdbcUrl, user, password);
GraphOracleSem graph = new GraphOracleSem(oracle, modelName);

Node nodeA = Node.createURI("http://A");
Node nodeB = Node.createURI("http://B");
Node nodeC = Node.createURI("http://C");
Node nodeD = Node.createURI("http://D");
Node nodeE = Node.createURI("http://E");
Node nodeF = Node.createURI("http://F");
Node nodeG = Node.createURI("http://G");
Node nodeX = Node.createURI("http://X");

// An anonymous node
Node ano = Node.createAnon(new AnonId("m1"));

Node relL = Node.createURI("http://likes");
Node relD = Node.createURI("http://dislikes");
Node relK = Node.createURI("http://knows");
Node relC = Node.createURI("http://differs");

graph.add(new Triple(nodeA, relL, nodeB));
graph.add(new Triple(nodeA, relC, nodeD));
graph.add(new Triple(nodeB, relL, nodeC));
graph.add(new Triple(nodeA, relD, nodeC));

graph.add(new Triple(nodeB, relD, ano));
graph.add(new Triple(nodeC, relL, nodeD));
graph.add(new Triple(nodeC, relK, nodeE));
graph.add(new Triple(ano,   relL, nodeD));
graph.add(new Triple(ano,   relL, nodeF));
graph.add(new Triple(ano,   relD, nodeB));

// X only likes itself
graph.add(new Triple(nodeX, relL, nodeX));

graph.commitTransaction();
HashMap<Node, Double> costMap = new HashMap<Node, Double>();
costMap.put(relL, Double.valueOf((double)0.5));
costMap.put(relD, Double.valueOf((double)1.5));
costMap.put(relC, Double.valueOf((double)5.5));

graph.setDOP(4); // this allows the underlying LINK/NODE tables
                 // and indexes to be created in parallel.

SemNetworkAnalyst sna = SemNetworkAnalyst.getInstance(
    graph,   // network data source
    true,    // directed graph
    true,    // cleanup existing NODE and LINK table
    costMap
    );

psOut.println("From nodeA to nodeC");
Node[] nodeArray = sna.shortestPathDijkstra(nodeA, nodeC);
printNodeArray(nodeArray, psOut);

psOut.println("From nodeA to nodeD");
nodeArray = sna.shortestPathDijkstra( nodeA, nodeD);
```

```
printNodeArray(nodeArray, psOut);

psOut.println("From nodeA to nodeF");
nodeArray = sna.shortestPathAStar(nodeA, nodeF);
printNodeArray(nodeArray, psOut);

psOut.println("From ano to nodeC");
nodeArray = sna.shortestPathAStar(ano, nodeC);
printNodeArray(nodeArray, psOut);

psOut.println("From ano to nodeX");
nodeArray = sna.shortestPathAStar(ano, nodeX);
printNodeArray(nodeArray, psOut);

graph.close();
oracle.dispose();
...
...

// A helper function to print out a path
public static void printNodeArray(Node[] nodeArray, PrintStream psOut)
{
  if (nodeArray == null) {
    psOut.println("Node Array is null");
    return;
  }
  if (nodeArray.length == 0) {psOut.println("Node Array is empty"); }
  int iFlag = 0;
  psOut.println("printNodeArray: full path starts");
  for (int iHops = 0; iHops < nodeArray.length; iHops++) {
    psOut.println("printNodeArray: full path item " + iHops + " = "
        + ((iFlag == 0) ? "[n] ":"[e] ") + nodeArray[iHops]);
    iFlag = 1 - iFlag;
  }
}
```

In Example 7-6:

- A `GraphOracleSem` object is constructed and a few triples are added to the `GraphOracleSem` object. These triples describe several individuals and their relationships including *likes*, *dislikes*, *knows*, and *differs*.
- A cost mapping is constructed to assign a numeric cost value to different links/predicates (of the RDF graph). In this case, 0.5, 1.5, and 5.5 are assigned to predicates *likes*, *dislikes*, and *differs*, respectively. This cost mapping is optional. If the mapping is absent, then all predicates will be assigned the same cost 1. When cost mapping is specified, this mapping does not need to be complete; for predicates not included in the cost mapping, a default value of 1 is assigned.

The output of Example 7-6 is as follows. In this output, the shortest paths are listed for the given start and end nodes. Note that the return value of `sna.shortestPathAStar(ano, nodeX)` is null because there is no path between these two nodes.

```
From nodeA to nodeC
printNodeArray: full path starts
printNodeArray: full path item 0 = [n] http://A          ## "n" denotes
Node
printNodeArray: full path item 1 = [e] http://likes      ## "e" denotes Edge (Link)
printNodeArray: full path item 2 = [n] http://B
printNodeArray: full path item 3 = [e] http://likes
printNodeArray: full path item 4 = [n] http://C
```

```
From nodeA to nodeD
printNodeArray: full path starts
printNodeArray: full path item 0 = [n] http://A
printNodeArray: full path item 1 = [e] http://likes
printNodeArray: full path item 2 = [n] http://B
printNodeArray: full path item 3 = [e] http://likes
printNodeArray: full path item 4 = [n] http://C
printNodeArray: full path item 5 = [e] http://likes
printNodeArray: full path item 6 = [n] http://D

From nodeA to nodeF
printNodeArray: full path starts
printNodeArray: full path item 0 = [n] http://A
printNodeArray: full path item 1 = [e] http://likes
printNodeArray: full path item 2 = [n] http://B
printNodeArray: full path item 3 = [e] http://dislikes
printNodeArray: full path item 4 = [n] m1
printNodeArray: full path item 5 = [e] http://likes
printNodeArray: full path item 6 = [n] http://F

From ano to nodeC
printNodeArray: full path starts
printNodeArray: full path item 0 = [n] m1
printNodeArray: full path item 1 = [e] http://dislikes
printNodeArray: full path item 2 = [n] http://B
printNodeArray: full path item 3 = [e] http://likes
printNodeArray: full path item 4 = [n] http://C

From ano to nodeX
Node Array is null
```

The underlying RDF graph view (SEMM_<*rdf_graph_name*> or RDFM_<*rdf_graph_name*>) cannot be used directly by NDM functions, and so `SemNetworkAnalyst` creates necessary tables that contain the nodes and links that are derived from a given RDF graph. These tables are not updated automatically when the RDF graph changes; rather, you can set the `cleanup` parameter in `SemNetworkAnalyst.getInstance` to `true`, to remove old node and link tables and to rebuild updated tables.

Example 7-7 Implementing NDM nearestNeighbors Function on Top of RDF Data

Example 7-7 implements the NDM `nearestNeighbors` function on top of RDF data. This gets a `NetworkAnalyst` object from the `SemNetworkAnalyst` instance, gets the node ID, creates `PointOnNet` objects, and processes `LogicalSubPath` objects.

```
%cat TestNearestNeighbor.java

import java.io.*;
import java.util.*;
import org.apache.jena.graph.*;
import org.apache.jena.update.*;
import oracle.spatial.rdf.client.jena.*;
import oracle.spatial.rdf.client.jena.SemNetworkAnalyst;
import oracle.spatial.network.lod.LODGoalNode;
import oracle.spatial.network.lod.LODNetworkConstraint;
import oracle.spatial.network.lod.NetworkAnalyst;
import oracle.spatial.network.lod.PointOnNet;
import oracle.spatial.network.lod.LogicalSubPath;

/**
 * This class implements a nearestNeighbors function on top of RDF data
 * using public APIs provided in SemNetworkAnalyst and Oracle Spatial NDM
 */
```

```
public class TestNearestNeighbor
{
  public static void main(String[] args) throws Exception
  {
    String szJdbcURL = args[0];
    String szUser    = args[1];
    String szPasswd  = args[2];

    PrintStream psOut = System.out;

    Oracle oracle = new Oracle(szJdbcURL, szUser, szPasswd);

    String szModelName = "test_nn";
    // First construct a TBox and load a few axioms
    ModelOracleSem model = ModelOracleSem.createOracleSemModel(oracle, szModelName);
    String insertString =
      " PREFIX my:  <http://my.com/> " +
      " INSERT DATA "                     +
      " { my:A   my:likes my:B .          " +
      "   my:A   my:likes my:C .          " +
      "   my:A   my:knows my:D .          " +
      "   my:A   my:dislikes my:X .       " +
      "   my:A   my:dislikes my:Y .       " +
      "   my:C   my:likes my:E .          " +
      "   my:C   my:likes my:F .          " +
      "   my:C   my:dislikes my:M .       " +
      "   my:D   my:likes my:G .          " +
      "   my:D   my:likes my:H .          " +
      "   my:F   my:likes my:M .          " +
      " }   ";
    UpdateAction.parseExecute(insertString,  model);

    GraphOracleSem g = model.getGraph();
    g.commitTransaction();
    g.setDOP(4);

    HashMap<Node, Double> costMap = new HashMap<Node, Double>();
    costMap.put(Node.createURI("http://my.com/likes"),    Double.valueOf(1.0));
    costMap.put(Node.createURI("http://my.com/dislikes"), Double.valueOf(4.0));
    costMap.put(Node.createURI("http://my.com/knows"),    Double.valueOf(2.0));

    SemNetworkAnalyst sna = SemNetworkAnalyst.getInstance(
        g,      // source RDF graph
        true,   // directed graph
        true,   // cleanup old Node/Link tables
        costMap
        );

    Node nodeStart = Node.createURI("http://my.com/A");
    long origNodeID = sna.getNodeID(nodeStart);

    long[] lIDs = {origNodeID};

    // translate from the original ID
    long nodeID = (sna.mapNodeIDs(lIDs))[0];

    NetworkAnalyst networkAnalyst = sna.getEmbeddedNetworkAnalyst();

    LogicalSubPath[] lsps = networkAnalyst.nearestNeighbors(
      new PointOnNet(nodeID),       // startPoint
      6,                            // numberOfNeighbors
      1,                            // searchLinkLevel
```

```
      1,                            // targetLinkLevel
      (LODNetworkConstraint) null, // constraint
      (LODGoalNode) null           // goalNodeFilter
      );

    if (lsps != null) {
      for (int idx = 0; idx < lsps.length; idx++) {
        LogicalSubPath lsp = lsps[idx];
        Node[] nodePath = sna.processLogicalSubPath(lsp, nodeStart);
        psOut.println("Path " + idx);
        printNodeArray(nodePath, psOut);
      }
    }

    g.close();
    sna.close();
    oracle.dispose();
  }


  public static void printNodeArray(Node[] nodeArray, PrintStream psOut)
  {
    if (nodeArray == null) {
      psOut.println("Node Array is null");
      return;
    }
    if (nodeArray.length == 0) {
      psOut.println("Node Array is empty");
    }
    int iFlag = 0;
    psOut.println("printNodeArray: full path starts");
    for (int iHops = 0; iHops < nodeArray.length; iHops++) {
      psOut.println("printNodeArray: full path item " + iHops + " = "
          + ((iFlag == 0) ? "[n] ":"[e] ") + nodeArray[iHops]);
      iFlag = 1 - iFlag;
    }
  }
}
```

The output of Example 7-7 is as follows.

```
Path 0
printNodeArray: full path starts
printNodeArray: full path item 0 = [n] http://my.com/A
printNodeArray: full path item 1 = [e] http://my.com/likes
printNodeArray: full path item 2 = [n] http://my.com/C

Path 1
printNodeArray: full path starts
printNodeArray: full path item 0 = [n] http://my.com/A
printNodeArray: full path item 1 = [e] http://my.com/likes
printNodeArray: full path item 2 = [n] http://my.com/B

Path 2
printNodeArray: full path starts
printNodeArray: full path item 0 = [n] http://my.com/A
printNodeArray: full path item 1 = [e] http://my.com/knows
printNodeArray: full path item 2 = [n] http://my.com/D

Path 3
printNodeArray: full path starts
printNodeArray: full path item 0 = [n] http://my.com/A
```

```
printNodeArray: full path item 1 = [e] http://my.com/likes
printNodeArray: full path item 2 = [n] http://my.com/C
printNodeArray: full path item 3 = [e] http://my.com/likes
printNodeArray: full path item 4 = [n] http://my.com/E

Path 4
printNodeArray: full path starts
printNodeArray: full path item 0 = [n] http://my.com/A
printNodeArray: full path item 1 = [e] http://my.com/likes
printNodeArray: full path item 2 = [n] http://my.com/C
printNodeArray: full path item 3 = [e] http://my.com/likes
printNodeArray: full path item 4 = [n] http://my.com/F

Path 5
printNodeArray: full path starts
printNodeArray: full path item 0 = [n] http://my.com/A
printNodeArray: full path item 1 = [e] http://my.com/knows
printNodeArray: full path item 2 = [n] http://my.com/D
printNodeArray: full path item 3 = [e] http://my.com/likes
printNodeArray: full path item 4 = [n] http://my.com/H
```


7.10.1 Generating Contextual Information about a Path in a Graph

It is sometimes useful to see contextual information about a path in a graph, in addition to the path itself. The `buildSurroundingSubGraph` method in the `SemNetworkAnalyst` class can output a DOT file (graph description language file, extension `.gv`) into the specified `Writer` object. For each node in the path, up to ten direct neighbors are used to produce a surrounding subgraph for the path. The following example shows the usage of generating a DOT file with contextual information, specifically the output from the analytical functions used in Example 7-6.

```
nodeArray = sna.shortestPathDijkstra(nodeA, nodeD);
printNodeArray(nodeArray, psOut);

FileWriter dotWriter = new FileWriter("Shortest_Path_A_to_D.gv");
sna.buildSurroundingSubGraph(nodeArray, dotWriter);
```

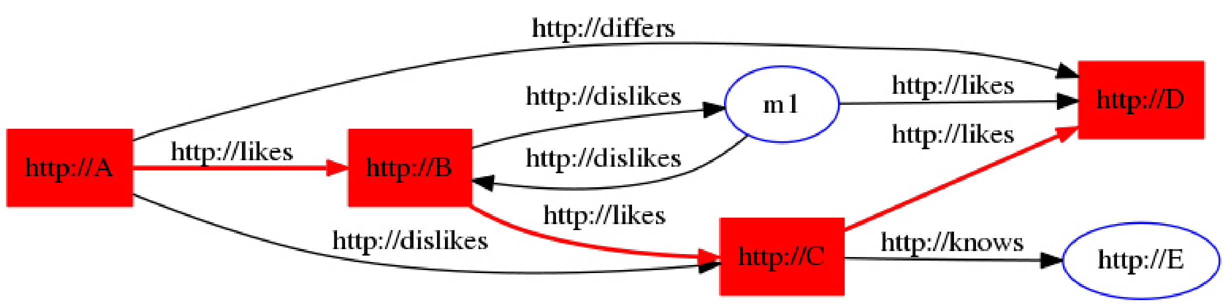
The generated output DOT file from the preceding example is straightforward, as shown in the following example:

```
% cat Shortest_Path_A_to_D.gv
digraph { rankdir = LR; charset="utf-8";

"Rhttp://A" [ label="http://A" shape=rectangle,color=red,style = filled, ];
"Rhttp://B" [ label="http://B" shape=rectangle,color=red,style = filled, ];
"Rhttp://A" -> "Rhttp://B" [ label="http://likes"  color=red, style=bold, ];
"Rhttp://C" [ label="http://C" shape=rectangle,color=red,style = filled, ];
"Rhttp://A" -> "Rhttp://C" [ label="http://dislikes" ];
"Rhttp://D" [ label="http://D" shape=rectangle,color=red,style = filled, ];
"Rhttp://A" -> "Rhttp://D" [ label="http://differs" ];
"Rhttp://B" -> "Rhttp://C" [ label="http://likes"  color=red, style=bold, ];
"Rm1" [ label="m1" shape=ellipse,color=blue, ];
"Rhttp://B" -> "Rm1" [ label="http://dislikes" ];
"Rm1" -> "Rhttp://B" [ label="http://dislikes" ];
"Rhttp://C" -> "Rhttp://D" [ label="http://likes"  color=red, style=bold, ];
"Rhttp://E" [ label="http://E" shape=ellipse,color=blue, ];
"Rhttp://C" -> "Rhttp://E" [ label="http://knows" ];
"Rm1" -> "Rhttp://D" [ label="http://likes" ];
}
```

You can also use methods in the `SemNetworkAnalyst` and `GraphOracleSem` classes to produce more sophisticated visualization of the analytical function output.

You can convert the preceding DOT file into a variety of image formats. Figure 7-1 is an image representing the information in the preceding DOT file.

Figure 7-1 Visual Representation of Analytical Function Output



7.11 Support for Server-Side APIs

This section describes some of the RDF Graph features that are exposed by RDF Graph support for Apache Jena.

For comprehensive documentation of the API calls that support the available features, see the RDF Graph support for Apache Jena reference information (Javadoc). For additional information about the server-side features exposed by the support for Apache Jena, see the relevant chapters in this manual.

- RDF Graph Collections Support
- Connection Pooling Support
- RDF Graph PL/SQL Interfaces
- Inference Options
- PelletInfGraph Class Support Deprecated

7.11.1 RDF Graph Collections Support

RDF graph collections (explained in RDF Graph Collections) are specified in the `GraphOracleSem` constructor, and they are handled transparently. If an RDF graph collection exists for the model-rulebase combination, it is used in query answering; if such an RDF graph collection does not exist, it is created in the database.

> **Note:**
>
> RDF graph collection support through the support for Apache Jena is available only with Oracle Database Release 11.2 or later.

The following example reuses an existing RDF graph collection.

```
String modelName = "EX";
String m1 = "EX_1";
```

```
ModelOracleSem defaultModel =
  ModelOracleSem.createOracleSemModel(oracle, modelName);

// create these RDF graphs in case they don't exist
ModelOracleSem model1 = ModelOracleSem.createOracleSemModel(oracle, m1);

String vmName = "VM_" + modelName;


//create an RDF graph collection containing EX and EX_1
oracle.executeCall(
"begin sem_apis.create_virtual_model(?,sem_models('"+ m1 + "','"+ modelName+"'),null);
end;",vmName);

String[] modelNames = {m1};
String[] rulebaseNames = {};

Attachment attachment = Attachment.createInstance(modelNames, rulebaseNames,
InferenceMaintenanceMode.NO_UPDATE, QueryOptions.ALLOW_QUERY_VALID_AND_DUP);

// vmName is passed to the constructor, so GraphOracleSem will use the RDF graph
collection
// named vmname (if the current user has read privileges on it)
GraphOracleSem graph = new GraphOracleSem(oracle, modelName, attachment, vmName);
graph.add(Triple.create(Node.createURI("urn:alice"),
                        Node.createURI("http://xmlns.com/foaf/0.1/mbox"),
                        Node.createURI("mailto:alice@example")));
ModelOracleSem model = new ModelOracleSem(graph);

String queryString =

   " SELECT ?subject ?object WHERE { ?subject ?p ?object } ";

Query query = QueryFactory.create(queryString) ;
QueryExecution qexec = QueryExecutionFactory.create(query, model) ;

try {
   ResultSet results = qexec.execSelect() ;
   for ( ; results.hasNext() ; ) {
      QuerySolution soln = results.nextSolution() ;
      psOut.println("soln " + soln);
   }
}
finally {
   qexec.close() ;
}

OracleUtils.dropSemanticModel(oracle, modelName);
OracleUtils.dropSemanticModel(oracle, m1);

oracle.dispose();
```

You can also use the GraphOracleSem constructor to create an RDF graph collection, as in the following example:

```
GraphOracleSem graph = new GraphOracleSem(oracle, modelName, attachment, true);
```

In this example, the fourth parameter (`true`) specifies that an RDF graph collection needs to be created for the specified `modelName` and `attachment`.

7.11.2 Connection Pooling Support

Oracle Database Connection Pooling is provided through the support for Apache Jena `OraclePool` class. Once this class is initialized, it can return Oracle objects out of its pool of available connections. Oracle objects are essentially database connection wrappers. After `dispose` is called on the Oracle object, the connection is returned to the pool. More information about using `OraclePool` can be found in the API reference information (Javadoc).

The following example sets up an OraclePool object with five (5) initial connections.

```
public static void main(String[] args) throws Exception
  {
    String szJdbcURL = args[0];
    String szUser    = args[1];
    String szPasswd  = args[2];
    String szModelName = args[3];

    // test with connection properties
    java.util.Properties prop = new java.util.Properties();
    prop.setProperty("MinLimit", "2");     // the cache size is 2 at least
    prop.setProperty("MaxLimit", "10");
    prop.setProperty("InitialLimit", "2"); // create 2 connections at startup
    prop.setProperty("InactivityTimeout", "1800");    //  seconds
    prop.setProperty("AbandonedConnectionTimeout", "900");  //  seconds
    prop.setProperty("MaxStatementsLimit", "10");
    prop.setProperty("PropertyCheckInterval", "60"); // seconds

    System.out.println("Creating OraclePool");
    OraclePool op = new OraclePool(szJdbcURL, szUser, szPasswd, prop,
               "OracleSemConnPool");
    System.out.println("Done creating OraclePool");

    // grab an Oracle and do something with it
    System.out.println("Getting an Oracle from OraclePool");
    Oracle oracle = op.getOracle();
    System.out.println("Done");
    System.out.println("Is logical connection:" +
        oracle.getConnection().isLogicalConnection());
    GraphOracleSem g = new GraphOracleSem(oracle, szModelName);
    g.add(Triple.create(Node.createURI("u:John"),
                        Node.createURI("u:parentOf"),
                        Node.createURI("u:Mary")));
    g.close();
    // return the Oracle back to the pool
    oracle.dispose();

    // grab another Oracle and do something else
    System.out.println("Getting an Oracle from OraclePool");
    oracle = op.getOracle();
    System.out.println("Done");
    System.out.println("Is logical connection:" +
        oracle.getConnection().isLogicalConnection());
    g = new GraphOracleSem(oracle, szModelName);
    g.add(Triple.create(Node.createURI("u:John"),
                        Node.createURI("u:parentOf"),
                        Node.createURI("u:Jack")));
    g.close();

    OracleUtils.dropSemanticModel(oracle, szModelName);
```

```
    // return the Oracle object back to the pool
    oracle.dispose();
}
```

7.11.3 RDF Graph PL/SQL Interfaces

Several RDF PL/SQL subprograms are available through the support for Apache Jena. Table 7-2 lists the subprograms and their corresponding Java class and methods.

Table 7-2 PL/SQL Subprograms and Corresponding RDF graph support for Apache Jena Java Class and Methods

PL/SQL Subprogram	Corresponding Java Class and Methods
SEM_APIS.DROP_RDF_GRAPH	OracleUtils.dropSemanticModel
SEM_APIS.MERGE_RDF_GRAPHS	OracleUtils.mergeModels
SEM_APIS.SWAP_NAMES	OracleUtils.swapNames
SEM_APIS.REMOVE_DUPLICATES	OracleUtils.removeDuplicates
SEM_APIS.RENAME_RDF_GRAPH	OracleUtils.renameModels

For information about these PL/SQL utility subprograms, see the reference information in SEM_APIS Package Subprograms. For information about the corresponding Java class and methods, see the RDF graph support for Apache Jena API Reference documentation (Javadoc).

7.11.4 Inference Options

You can add options to inferred graph calls by using the following methods in the `Attachment` class (in package `oracle.spatial.rdf.client.jena`):

```
public void setUseLocalInference(boolean useLocalInference)
public boolean getUseLocalInference()

public void setDefGraphForLocalInference(String defaultGraphName)
public String getDefGraphForLocalInference()

public String getInferenceOption()
public void setInferenceOption(String inferenceOption)
```

Example 7-8 Specifying Inference Options

For more information about these methods, see the Javadoc.

Example 7-8 enables parallel inference (with a degree of 4) and RAW format when creating an inferred graph. The example also uses the `performInference` method to create the inferred graph (comparable to using the SEM_APIS.CREATE_INFERRED_GRAPH PL/SQL procedure).

```
import java.io.*;
import org.apache.jena.query.*;
import oracle.spatial.rdf.client.jena.*;
import org.apache.jena.update.*;
import org.apache.jena.sparql.core.DatasetImpl;

public class TestNewInference
{
  public static void main(String[] args) throws Exception
```

```
  {
    String szJdbcURL = args[0];
    String szUser    = args[1];
    String szPasswd  = args[2];

    PrintStream psOut = System.out;

    Oracle oracle = new Oracle(szJdbcURL, szUser, szPasswd);

    String szTBoxName = "test_new_tbox";
    {
      // First construct a TBox and load a few axioms
      ModelOracleSem modelTBox = ModelOracleSem.createOracleSemModel(oracle, szTBoxName);
      String insertString =
        " PREFIX my:  <http://my.com/> " +
        " PREFIX rdfs:  <http://www.w3.org/2000/01/rdf-schema#> " +
        " INSERT DATA "                                        +
        " { my:C1  rdfs:subClassOf my:C2 .                   " +
        "   my:C2  rdfs:subClassOf my:C3 .                   " +
        "   my:C3  rdfs:subClassOf my:C4 .                   " +
        " }   ";
      UpdateAction.parseExecute(insertString,  modelTBox);
      modelTBox.close();
    }

    String szABoxName = "test_new_abox";
    {
      // Construct an ABox and load a few quads
      ModelOracleSem modelABox = ModelOracleSem.createOracleSemModel(oracle, szABoxName);
      DatasetGraphOracleSem dataset =
DatasetGraphOracleSem.createFrom(modelABox.getGraph());
      modelABox.close();

      String insertString =
        " PREFIX my:    <http://my.com/> " +
        " PREFIX rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
        " INSERT DATA                                                 " +
        " { GRAPH my:G1 { my:I1  rdf:type my:C1 .                     " +
        "                 my:I2  rdf:type my:C2 .                     " +
        "               }                                             " +
        " };                                                          " +
        " INSERT DATA                                                 " +
        " { GRAPH my:G2 { my:J1  rdf:type my:C3 .                     " +
        "               }                                             " +
        " }    ";
      UpdateAction.parseExecute(insertString,  dataset);
      dataset.close();
    }


    String[] attachedModels = new String[1];
    attachedModels[0] = szTBoxName;

    String[] attachedRBs = {"OWL2RL"};

    Attachment attachment = Attachment.createInstance(
        attachedModels, attachedRBs,
        InferenceMaintenanceMode.NO_UPDATE,
        QueryOptions.ALLOW_QUERY_INVALID);

    // We are going to run named graph based local inference
    attachment.setUseLocalInference(true);
```

```
    // Set the default graph (TBox)
    attachment.setDefGraphForLocalInference(szTBoxName);

    // Set the inference option to use parallel inference
    // with a degree of 4, and RAW8 format.
    attachment.setInferenceOption("DOP=4,RAW8=T");

    GraphOracleSem graph = new GraphOracleSem(
        oracle,
        szABoxName,
        attachment
        );
    DatasetGraphOracleSem dsgos = DatasetGraphOracleSem.createFrom(graph);
    graph.close();

    // Invoke create_inferred_graph PL/SQL API
    dsgos.performInference();

    psOut.println("TestNewInference: # of inferred graph " +
        Long.toString(dsgos.getInferredGraphSize()));

    String queryString =
      " SELECT ?g ?s ?p ?o WHERE {  GRAPH ?g {?s ?p ?o } } " ;

    Query query = QueryFactory.create(queryString, Syntax.syntaxARQ);
    QueryExecution qexec = QueryExecutionFactory.create(
        query, DatasetImpl.wrap(dsgos));
    ResultSet results = qexec.execSelect();

    ResultSetFormatter.out(psOut, results);

    dsgos.close();
    oracle.dispose();
  }
}
```

The output of Example 7-8 is as follows.

```
TestNewInference: # of inferred graph 9

-----------------------------------------------------------------------------------------
---------------------------
| g                    | s                    |
p                                                  | o                    |
=========================================================================================
===========================
| <http://my.com/G1> | <http://my.com/I2> | <http://www.w3.org/1999/02/22-rdf-syntax-
ns#type> | <http://my.com/C3> |
| <http://my.com/G1> | <http://my.com/I2> | <http://www.w3.org/1999/02/22-rdf-syntax-
ns#type> | <http://my.com/C2> |
| <http://my.com/G1> | <http://my.com/I2> | <http://www.w3.org/1999/02/22-rdf-syntax-
ns#type> | <http://my.com/C4> |
| <http://my.com/G1> | <http://my.com/I1> | <http://www.w3.org/1999/02/22-rdf-syntax-
ns#type> | <http://my.com/C3> |
| <http://my.com/G1> | <http://my.com/I1> | <http://www.w3.org/1999/02/22-rdf-syntax-
ns#type> | <http://my.com/C1> |
| <http://my.com/G1> | <http://my.com/I1> | <http://www.w3.org/1999/02/22-rdf-syntax-
ns#type> | <http://my.com/C2> |
| <http://my.com/G1> | <http://my.com/I1> | <http://www.w3.org/1999/02/22-rdf-syntax-
ns#type> | <http://my.com/C4> |
| <http://my.com/G2> | <http://my.com/J1> | <http://www.w3.org/1999/02/22-rdf-syntax-
```

```
ns#type> | <http://my.com/C3> |
| <http://my.com/G2> | <http://my.com/J1> | <http://www.w3.org/1999/02/22-rdf-syntax-
ns#type> | <http://my.com/C4> |
-----------------------------------------------------------------------------------------
---------------------------
```

For information about using OWL inferencing, see Using OWL Inferencing.

7.11.5 PelletInfGraph Class Support Deprecated

The support for the `PelletInfGraph` class within the support for Apache Jena is deprecated. You should instead use the more optimized Oracle/Pellet integration through the PelletDb OWL 2 reasoner for Oracle Database.

7.12 Bulk Loading Using RDF Graph Support for Apache Jena

To load thousands to hundreds of thousands of RDF/OWL data files into an Oracle database, you can use the `prepareBulk` and `completeBulk` methods in the `OracleBulkUpdateHandler` Java class to simplify the task.

The `addInBulk` method in the `OracleBulkUpdateHandler` class can load triples of an RDF graph into an Oracle database in bulk loading style. If the graph is a Jena in-memory graph, the operation is limited by the size of the physical memory. The `prepareBulk` method bypasses the Jena in-memory graphs and takes a direct input stream to an RDF data file, parses the data, and load the triples into an underlying staging table. If the staging table and an accompanying table for storing long literals do not already exist, they are created automatically.

The `prepareBulk` method can be invoked multiple times to load multiple data files into the same underlying staging table. It can also be invoked concurrently, assuming the hardware system is balanced and there are multiple CPU cores and sufficient I/O capacity.

Once all the data files are processed by the `prepareBulk` method, you can invoke `completeBulk` to load all the data into the RDF network.

Example 7-9 Loading Data into the Staging Table (prepareBulk)

Example 7-9 shows how to load all data files in directory `dir_1` into the underlying staging table. Long literals are supported and will be stored in a separate table. The data files can be compressed using GZIP to save storage space, and the `prepareBulk` method can detect automatically if a data file is compressed using GZIP or not.

```
Oracle oracle = new Oracle(szJdbcURL, szUser, szPasswd);
GraphOracleSem graph = new GraphOracleSem(oracle, szModelName);

PrintStream psOut = System.out;

String dirname = "dir_1";
File fileDir = new File(dirname);
String[] szAllFiles = fileDir.list();

// loop through all the files in a directory
for (int idx = 0; idx < szAllFiles.length; idx++) {
  String szIndFileName = dirname + File.separator + szAllFiles[idx];
  psOut.println("process to [ID = " + idx + " ] file " + szIndFileName);
  psOut.flush();

  try {
    InputStream is = new FileInputStream(szIndFileName);
    graph.getBulkUpdateHandler().prepareBulk(
```

```
        is,                    // input stream
        "http://example.com",  // base URI
        "RDF/XML",             // data file type: can be RDF/XML, N-TRIPLE, etc.
        "SEMTS",               // tablespace
        null,                  // flags
        null,                  // listener
        null                   // staging table name.
        );
      is.close();
    }
    catch (Throwable t) {
      psOut.println("Hit exception " + t.getMessage());
    }
  }

graph.close();
oracle.dispose();
```

The code in Example 7-9, starting from creating a new Oracle object and ending with disposing of the Oracle object, can be executed in parallel. Assume there is a quad-core CPU and enough I/O capacity on the database hardware system; you can divide up all the data files and save them into four separate directories: `dir_1`, `dir_2`, `dir_3`, and `dir_4`. Four Java threads of processes can be started to work on those directories separately and concurrently. (For more information, see Using prepareBulk in Parallel (Multithreaded) Mode.)

Example 7-10 Loading Data from the Staging Table into the RDF Network (completeBulk)

After all data files are processed, you can invoke, just once, the `completeBulk` method to load the data from staging table into the RDF network, as shown in Example 7-10. Triples with long literals will be loaded also.

```
graph.getBulkUpdateHandler().completeBulk(
  null,  // flags for invoking SEM_APIS.bulk_load_from_staging_table
  null   // staging table name
);
```

The `prepareBulk` method can also take a Jena RDF graph as an input data source, in which case triples in that Jena RDF graph are loaded into the underlying staging table. For more information, see the Javadoc.

Example 7-11 Using prepareBulk with RDFa

In addition to loading triples from Jena RDF graphs and data files, the `prepareBulk` method supports RDFa, as shown in Example 7-11. (RDFa is explained in `http://www.w3.org/TR/xhtml-rdfa-primer/`.)

```
graph.getBulkUpdateHandler().prepareBulk(
  rdfaUrl,   // url to a web page using RDFa
  "SEMTS",   // tablespace
  null,      // flags
  null,      // listener
  null       // staging table name
);
```

To parse RDFa, the relevant `java-rdfa` libraries must be included in the classpath. No additional setup or API calls are required. (For information about java-rdfa, see `http://www.rootdev.net/maven/projects/java-rdfa/` and the other topics there under Project Information.)

Note that if the `rdfaUrl` is located outside a firewall, you may need to set the following HTTP Proxy-related Java VM properties:

```
-Dhttp.proxyPort=...
-Dhttp.proxyHost=...
```

Example 7-12 Loading Quads into a DatasetGraph

The preceding examples in this section load triple data into a single graph. Loading quad data that may span across multiple named graphs (such as data in NQUADS format) requires the use of the `DatasetGraphOracleSem` class. The `DatasetGraphOracleSem` class does not use the `BulkUpdateHandler` API, but does provide a similar `prepareBulk` and `completeBulk` interface, as shown in Example 7-12.

```
Oracle oracle = new Oracle(szJdbcURL, szUser, szPasswd);

// Can only create DatasetGraphOracleSem from an existing GraphOracleSem
GraphOracleSem graph = new GraphOracleSem(oracle, szModelName);
DatasetGraphOracleSem dataset = DatasetGraphOracleSem.createFrom(graph);

// Don't need graph anymore, close it to free resources
graph.close();

try {
    InputStream is = new FileInputStream(szFileName);
    // load NQUADS file into a staging table. This file can be gzipp'ed.
    dataset.prepareBulk(
        is,                  // input stream
        "http://my.base/",   // base URI
        "N-QUADS",           // data file type; can be "TRIG"
        "SEMTS",             // tablespace
        null,                // flags
        null,                // listener
        null,                // staging table name
        false                // truncate staging table before load
    );
    // Load quads from staging table into the dataset
    dataset.completeBulk(
        null, // flags; can be "PARSE PARALLEL_CREATE_INDEX PARALLEL=4
              //                mbv_method=shadow" on a quad core machine
        null  // staging table name
    );
}
catch (Throwable t) {
    System.out.println("Hit exception " + t.getMessage());
}
finally {
    dataset.close();
    oracle.dispose();
}
```

- Using prepareBulk in Parallel (Multithreaded) Mode
- Handling Illegal Syntax During Data Loading

7.12.1 Using prepareBulk in Parallel (Multithreaded) Mode

Example 7-9 provided a way to load, sequentially, a set of files under a file system directory to an Oracle Database table (staging table). Example 7-13 loads, concurrently, a set of files to an Oracle table (staging table). The degree of parallelism is controlled by the input parameter `iMaxThreads`.

On a balanced hardware setup with 4 or more CPU cores, setting `iMaxThreads` to 8 (or 16) can improve significantly the speed of `prepareBulk` operation when there are many data files to be processed.

Example 7-13   Using prepareBulk with iMaxThreads

```
public void testPrepareInParallel(String jdbcUrl, String user,
                       String password, String modelName,
                       String lang,
                       String tbs,
                       String dirname,
                       int iMaxThreads,
                       PrintStream psOut)
   throws SQLException, IOException,  InterruptedException
 {
   File dir = new File(dirname);
   File[] files = dir.listFiles();

   // create a set of physical Oracle connections and graph objects
   Oracle[] oracles = new Oracle[iMaxThreads];
   GraphOracleSem[] graphs = new GraphOracleSem[iMaxThreads];
   for (int idx = 0; idx < iMaxThreads; idx++) {
     oracles[idx] = new Oracle(jdbcUrl, user, password);
     graphs[idx] = new GraphOracleSem(oracles[idx], modelName);
   }

   PrepareWorker[] workers = new PrepareWorker[iMaxThreads];
   Thread[] threads = new Thread[iMaxThreads];
   for (int idx = 0; idx < iMaxThreads; idx++) {
     workers[idx] = new PrepareWorker(
         graphs[idx],
         files,
         idx,
         iMaxThreads,
         lang,
         tbs,
         psOut
         );
     threads[idx] = new Thread(workers[idx], workers[idx].getName());
     psOut.println("testPrepareInParallel: PrepareWorker " + idx + " running");
     threads[idx].start();
   }

   psOut.println("testPrepareInParallel: all threads started");

   for (int idx = 0; idx < iMaxThreads; idx++) {
     threads[idx].join();
   }
   for (int idx = 0; idx < iMaxThreads; idx++) {
     graphs[idx].close();
     oracles[idx].dispose();
   }
 }

 static class PrepareWorker implements Runnable
 {
   GraphOracleSem graph = null;
   int idx;
   int threads;
   File[] files = null;
   String lang = null;
   String tbs  = null;
```

```
    PrintStream psOut;

    public void run()
    {
      long lStartTime = System.currentTimeMillis();
      for (int idxFile = idx; idxFile < files.length; idxFile += threads) {
        File file = files[idxFile];
        try {
          FileInputStream fis = new FileInputStream(file);
          graph.getBulkUpdateHandler().prepareBulk(
              fis,
              "http://base.com/",
              lang,
              tbs,
              null,                   // flags
              new MyListener(psOut),  // listener
              null                    // table name
              );
          fis.close();
        }
        catch (Exception e) {
          psOut.println("PrepareWorker: thread ["+getName()+"] error "+ e.getMessage());
        }
        psOut.println("PrepareWorker: thread ["+getName()+"] done to "
            + idxFile + ", file = " + file.toString()
            + " in (ms) " + (System.currentTimeMillis() - lStartTime));
      }
    }

    public PrepareWorker(GraphOracleSem graph,
                         File[] files,
                         int idx,
                         int threads,
                         String lang,
                         String tbs,
                         PrintStream psOut)
    {
      this.graph   = graph;
      this.files   = files;
      this.psOut   = psOut;
      this.idx     = idx;
      this.threads = threads;
      this.files   = files;
      this.lang    = lang;
      this.tbs     = tbs ;
    }

    public String getName()
    {
      return "PrepareWorker" + idx;
    }
  }

  static class MyListener implements StatusListener
  {
    PrintStream m_ps = null;
    public MyListener(PrintStream ps) { m_ps = ps; }
    long lLastBatch = 0;

    public void statusChanged(long count)
    {
      if (count - lLastBatch >= 10000) {
```

```
        m_ps.println("process to " + Long.toString(count));
        lLastBatch = count;
      }
    }

    public int illegalStmtEncountered(Node graphNode, Triple triple, long count)
    {
      m_ps.println("hit illegal statement with object " + triple.getObject().toString());
      return 0; // skip it
    }
  }
```

7.12.2 Handling Illegal Syntax During Data Loading

You can skip illegal triples and quads when using `prepareBulk`. This feature is useful if the source RDF data may contain syntax errors. In Example 7-14, a customized implementation of the `StatusListener` interface (defined in package `oracle.spatial.rdf.client.jena`) is passed as a parameter to prepareBulk. In this example, the `illegalStmtEncountered` method prints the object field of the illegal triple, and returns 0 so that `prepareBulk` can skip that illegal triple and move on.

Example 7-14 Skipping Triples with Illegal Syntax

```
....

Oracle oracle = new Oracle(jdbcUrl, user, password);
GraphOracleSem graph = new GraphOracleSem(oracle, modelName);
PrintStream psOut = System.err;

graph.getBulkUpdateHandler().prepareBulk(
  new FileInputStream(rdfDataFilename),
  "http://base.com/",    // base
  lang,                  // data format, can be "N-TRIPLES" "RDF/XML" ...
  tbs,                   // tablespace name
  null,                  // flags
  new MyListener(psOut), // call back to show progress and also process illegal triples/
quads
  null,                  // tableName, if null use default names
  false                  // truncate existing staging tables
  );

 graph.close();
 oracle.dispose();
 ....

 // A customized StatusListener interface implementation
  public class MyListener implements StatusListener
 {
   PrintStream m_ps = null;
   public MyListener(PrintStream ps) { m_ps = ps; }

   public void statusChanged(long count)
   {
     // m_ps.println("process to " + Long.toString(count));
   }

  public int illegalStmtEncountered(Node graphNode, Triple triple, long count)
  {
    m_ps.println("hit illegal statement with object " + triple.getObject().toString());
    return 0; // skip it
```

```
  }
}
```

7.13 Automatic Variable Renaming

Automatic variable renaming can enable certain queries that previously failed to run successfully.

Previously, variable names used in SPARQL queries were passed directly on to Oracle Database as a part of a SQL statement. If the variable names included a SQL or PL/SQL reserved keyword, the query failed to execute. For example, the following SPARQL query used to fail because the word date as a special meaning to the Oracle Database SQL processing engine.

```
select ?date { :event  :happenedOn ?date }
```

Currently, this query does not fail, because a "smart scan" is performed and automatic replacement is done on certain reserved variable names (or variable names that are very long) before the query is sent to Oracle database for execution. The replacement is based on a list of reserved keywords that are stored in the following file embedded in `sdordfclient.jar`:

```
oracle/spatial/rdf/client/jena/oracle_sem_reserved_keywords.lst
```

This file contains over 100 entries, and you can edit the file to add entries if necessary.

The following are examples of SPARQL queries that use SQL or PL/SQL reserved keywords as variables, and that will succeed because of automatic variable renaming:

- Query using `SELECT` as a variable name:

  ```
  PREFIX foaf: <http://xmlns.com/foaf/0.1/>
  select ?SELECT ?z
  where
  {     ?SELECT foaf:name ?y.
        optional {?SELECT foaf:knows ?z.}
  }
  ```

- Query using `ARRAY` and `DATE` as variable names:

  ```
  PREFIX x:    <http://example.com#>
  construct {
      ?ARRAY x:date ?date .
  }
  where {
      ?ARRAY x:happenedOn ?date .
  }
  ```

7.14 JavaScript Object Notation (JSON) Format Support

JavaScript Object Notation (JSON) format is supported for SPARQL query responses. JSON data format is simple, compact, and well suited for JavaScript programs.

For example, assume the following Java code snippet, which calls the `ResultSetFormatter.outputAsJSON` method:

```
Oracle oracle = new Oracle(jdbcUrl, user, password);

GraphOracleSem graph = new GraphOracleSem(oracle, modelName);
ModelOracleSem model = new ModelOracleSem(graph);
```

```
graph.add(new Triple(
                   Node.createURI("http://ds1"),
                   Node.createURI("http://dp1"),
                   Node.createURI("http://do1")
                   )
         );

graph.add(new Triple(
                   Node.createURI("http://ds2"),
                   Node.createURI("http://dp2"),
                   Node.createURI("http://do2")
                   )
         );
graph.commitTransaction();

Query q = QueryFactory.create("select ?s ?p ?o where {?s ?p ?o}",
                              Syntax.syntaxARQ);
QueryExecution qexec = QueryExecutionFactory.create(q, model);

ResultSet results = qexec.execSelect();
ResultSetFormatter.outputAsJSON(System.out, results);
```

The JSON output is as follows:

```
{
  "head": {
    "vars": [ "s" , "p" , "o" ]
  } ,
  "results": {
    "bindings": [
      {
        "s": { "type": "uri" , "value": "http://ds1" } ,
        "p": { "type": "uri" , "value": "http://dp1" } ,
        "o": { "type": "uri" , "value": "http://do1" }
      } ,
      {
        "s": { "type": "uri" , "value": "http://ds2" } ,
        "p": { "type": "uri" , "value": "http://dp2" } ,
        "o": { "type": "uri" , "value": "http://do2" }
      }
    ]
  }
}
```

The preceding example can be changed as follows to query a remote SPARQL endpoint instead of directly against an Oracle database. (If the remote SPARQL endpoint is outside a firewall, then the HTTP Proxy probably needs to be set.)

```
Query q = QueryFactory.create("select ?s ?p ?o where {?s ?p ?o}",
                              Syntax.syntaxARQ);
QueryExecution qe = QueryExecutionFactory.sparqlService(sparqlURL, q);

ResultSet results = qexec.execSelect();
ResultSetFormatter.outputAsJSON(System.out, results);
```

To extend the first example in this section to named graphs, the following code snippet adds two quads to the same Oracle model, executes a named graph-based SPARQL query, and serializes the query output into JSON format:

```
DatasetGraphOracleSem dsgos = DatasetGraphOracleSem.createFrom(graph);
graph.close();
```

```
dsgos.add(new Quad(Node.createURI("http://g1"),
                   Node.createURI("http://s1"),
                   Node.createURI("http://p1"),
                   Node.createURI("http://o1")
                   )
         );
dsgos.add(new Quad(Node.createURI("http://g2"),
                   Node.createURI("http://s2"),
                   Node.createURI("http://p2"),
                   Node.createURI("http://o2")
                   )
         );

Query q1 = QueryFactory.create(
  "select ?g ?s ?p ?o where { GRAPH ?g {?s ?p ?o} }");

QueryExecution qexec1 = QueryExecutionFactory.create(q1,
    DatasetImpl.wrap(dsgos));

ResultSet results1 = qexec1.execSelect();
ResultSetFormatter.outputAsJSON(System.out, results1);

dsgos.close();
oracle.dispose();
```

The JSON output is as follows:


```
{
  "head": {
    "vars": [ "g" , "s" , "p" , "o" ]
  } ,
  "results": {
    "bindings": [
      {
        "g": { "type": "uri" , "value": "http://g1" } ,
        "s": { "type": "uri" , "value": "http://s1" } ,
        "p": { "type": "uri" , "value": "http://p1" } ,
        "o": { "type": "uri" , "value": "http://o1" }
      } ,
      {
        "g": { "type": "uri" , "value": "http://g2" } ,
        "s": { "type": "uri" , "value": "http://s2" } ,
        "p": { "type": "uri" , "value": "http://p2" } ,
        "o": { "type": "uri" , "value": "http://o2" }
      }
    ]
  }
}
```


You can also get a JSON response through HTTP against a Fuseki-based SPARQL endpoint, as in the following example. Normally, when executing a SPARQL query against a SPARQL Web service endpoint, the `Accept request-head` field is set to be `application/sparql-results+xml`. For JSON output format, replace the `Accept request-head` field with `application/sparql-results+json`.

```
http://hostname:7001/fuseki/oracle?query=<URL_ENCODED_SPARQL_QUERY>&output=json
```

7.15 Other Recommendations and Guidelines

This section contains various recommendations and other information related to SPARQL queries.

- BOUND or !BOUND Instead of EXISTS or NOT EXISTS
- SPARQL 1.1 SELECT Expressions
- Syntax Involving Bnodes (Blank Nodes)
- Limit in the SERVICE Clause
- OracleGraphWrapperForOntModel Class for Better Performance

7.15.1 BOUND or !BOUND Instead of EXISTS or NOT EXISTS

For better performance, use `BOUND` or `!BOUND` instead of `EXISTS` or `NOT EXISTS`.

7.15.2 SPARQL 1.1 SELECT Expressions

You can use SPARQL 1.1 SELECT expressions without any significant performance overhead, even if the function is not currently supported within Oracle Database. Examples include the following:

```
-- Query using SHA1 function
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX xsd:  <http://www.w3.org/2001/XMLSchema#>
PREFIX eg:   <http://biometrics.example/ns#>
SELECT ?name ?email (sha1(?email) as ?sha1)
WHERE
{
  ?x foaf:name  ?name ; eg:email ?email .
}

-- Query using CONCAT function
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>
SELECT ( CONCAT(?G, " ", ?S) AS ?name )
WHERE
{
  ?P foaf:givenName ?G ; foaf:surname ?S
}
```

7.15.3 Syntax Involving Bnodes (Blank Nodes)

Syntax involving bnodes can be used freely in query patterns. For example, the following bnode-related syntax is supported at the parser level, so each is equivalent to its full triple-query-pattern-based version.

```
:x :q [ :p "v" ] .

(1 ?x 3 4) :p "w" .

(1 [:p :q] ( 2 ) ) .
```

7.15.4 Limit in the SERVICE Clause

When writing a SPARQL 1.1 federated query, you can set a limit on returned rows in the subquery inside the SERVICE clause. This can effectively constrain the amount of data to be transported between the local repository and the remote SPARQL endpoint.

For example, the following query specifies `limit 100` in the subquery in the SERVICE clause:

```
PREFIX : <http://example.com/>
SELECT ?s ?o
```

```
WHERE
    {
      ?s :name "CA"
      SERVICE <http://REMOTE_SPARQL_ENDPOINT_HERE>
         {
            select ?s  ?o
                {?s :info ?o}
             limit 100
          }
    }
```

7.15.5 OracleGraphWrapperForOntModel Class for Better Performance

The Jena `OntModel` class lets you create, modify, and analyze an ontology stored in a Jena model. However, the `OntModel` implementation is not optimized for RDF data stored in a database. This results in suboptimal performance when using `OntModel` with an Oracle model. Therefore, the class `OracleGraphWrapperForOntModel` has been created to alleviate this performance issue.

The `OracleGraphWrapperForOntModel` class implements the Jena `Graph` interface and represents a graph backed by an Oracle RDF/OWL graph that is meant for use with the Jena `OntModel` API. The `OracleGraphWrapperForOntModel` class uses two RDF stores in a hybrid approach for persisting changes and responding to queries. Both RDF stores contain the same data, but one resides in memory while the other resides in the Oracle database.

When queried through `OntModel`, the `OracleGraphWrapperForOntModel` graph runs the queries against the in-memory store to improve performance. However, the `OracleGraphWrapperForOntModel` class persists changes made through `OntModel`, such as adding or removing classes, by applying changes to both stores.

Due to its hybrid approach, an `OracleGraphWrapperForOntModel` graph requires that sufficient memory be allocated to the JVM to store a copy of the ontology in memory. In internal experiments, it was found that an ontology with approximately 3 million triples requires 6 or more GB of physical memory.

Example 7-15   Using OntModel with Ontology Stored in Oracle Database

Example 7-15 shows how to use the `OntModel` APIs with an existing ontology stored in an Oracle model.

```
// Set up connection to Oracle RDF store and the Oracle model
// containing the ontology
Oracle oracle = new Oracle(szJdbcURL, szUser, szPasswd);
GraphOracleSem oracleGraph = new GraphOracleSem(oracle, szModelName);

// Create a new hybrid graph using the oracle graph to persist
// changes.  This method will copy all the data from the oracle graph
// into an in-memory graph, which may significantly increase JVM memory
// usage.
Graph hybridGraph = OracleGraphWrapperForOntModel.getInstance(oracleGraph);

// Build a model around the hybrid graph and wrap the model with Jena's
// OntModel
Model model = ModelFactory.createModelForGraph(hybridGraph);
OntModel ontModel = ModelFactory.createOntologyModel(ontModelSpec, model);

// Perform operations on the ontology
OntClass personClass = ontModel.createClass("<http://someuri/person>");
ontModel.createIndividual(personClass);
```

```
// Close resources (will also close oracleGraph)!
hybridGraph.close();
ontModel.close();
```

Note that any `OntModel` object created using `OracleGraphWrapperForOntModel` will not reflect changes made to the underlying Oracle model by another process, through a separate `OntModel`, or through a separate Oracle graph referencing the same underlying model. All changes to an ontology should go through a single `OntModel` object and its underlying `OracleGraphWrapperForOntModel` graph until the graph have been closed.

Example 7-16 Using a Custom In-Memory Graph

If the default in-memory RDF store used by `OracleGraphWrapperForOntModel` is not sufficient for an ontology and system, the class provides an interface for specifying a custom graph to use as the in-memory store. Example 7-16 shows how to create an `OracleGraphWrapperForOntModel` that uses a custom in-memory graph to answer queries from `OntModel`.

```
// Set up connection to Oracle RDF store and the Oracle model
// containing the ontology
Oracle oracle = new Oracle(szJdbcURL, szUser, szPasswd);
GraphOracleSem oracleGraph = new GraphOracleSem(oracle, szModelName);

// Create a custom in-memory graph to use instead of the default
// Jena in-memory graph for quickly answering OntModel queries.
// Note that this graph does not *need* to be in-memory, but in-memory
// is preferred.
GraphBase queryGraph = new CustomInMemoryGraphImpl();

// Create a new hybrid graph using the oracle graph to persist
// changes and the custom in-memory graph to answer queries.
// Also set the degree of parallelism to use when copying data from
// the oracle graph to the querying graph.
int degreeOfParallelism = 4;
Graph hybridGraph = OracleGraphWrapperForOntModel.getInstance(oracleGraph, queryGraph,
degreeOfParallelism);

// Build a model and wrap the model with Jena's OntModel
Model model = ModelFactory.createModelForGraph(hybridGraph);
OntModel ontModel = ModelFactory.createOntologyModel(ontModelSpec, model);

// Perform operations on the ontology
// ...

// Close resources (will close oracleGraph and queryGraph)!
hybridGraph.close();
ontModel.close();
```

7.16 Example Queries Using RDF Graph Support for Apache Jena

This section includes example queries using the support for Apache Jena. Each example is self-contained: it typically creates a model, creates triples, performs a query that may involve inference, displays the result, and drops the RDF graph.

The example queries perform the following:

- Count asserted triples and asserted plus inferred triples in an example "university" ontology, both by referencing the ontology by a URL and by bulk loading the ontology from a local file.
- Run several SPARQL queries using a "family" ontology, including features such as LIMIT, OFFSET, TIMEOUT, DOP (degree of parallelism), ASK, DESCRIBE, CONSTRUCT, GRAPH, ALLOW_DUP (duplicate triples with multiple models), SPARUL (inserting data)
- Use the ARQ built-in function
- Use a SELECT cast query
- Instantiate Oracle Database using OracleConnection
- Use Oracle Database connection pooling

To run a query, you must do the following:

1. Include the code in a Java source file. The examples used in this section are supplied in files in the `examples` directory of the support for Apache Jena download.
2. Compile the Java source file. For example:

   ```
   > javac -classpath ../jar/'*' Test.java
   ```

   > **Note:**
   >
   > The `javac` and `java` commands must each be on a single command line.

3. Run the compiled file. For example:

   ```
   java -classpath ./:../jar/'*' Test jdbc:oracle:thin:@localhost:1521:orcl scott
   <password-for-scott> TestModel NET1
   ```

- Query Family Relationships
- Load OWL Ontology and Perform OWLPrime Inference
- Bulk Load OWL Ontology and Perform OWLPrime Inference
- SPARQL OPTIONAL Query
- SPARQL Query with LIMIT and OFFSET
- SPARQL Query with TIMEOUT and DOP
- Query Involving Named Graphs
- SPARQL ASK Query
- SPARQL DESCRIBE Query
- SPARQL CONSTRUCT Query
- Query Multiple Models and Specify "Allow Duplicates"
- SPARQL Update
- SPARQL Query with ARQ Built-In Functions
- SELECT Cast Query
- Instantiate Oracle Database Using OracleConnection
- Oracle Database Connection Pooling

7.16.1 Query Family Relationships

Example 7-17 Query Family Relationships

The following example specifies that John is the father of Mary, and it selects and displays the subject and object in each `fatherOf` relationship

```
import oracle.spatial.rdf.client.jena.*;
import org.apache.jena.rdf.model.Model;
import org.apache.jena.graph.*;
import org.apache.jena.query.*;

public class Test_privnet {

  public static void main(String[] args) throws Exception
  {

    String szJdbcURL = args[0];
    String szUser    = args[1];
    String szPasswd  = args[2];

    String szModelName = args[3];
    String szNetworkName = args[4];

    Oracle oracle = new Oracle(szJdbcURL, szUser, szPasswd);
    Model model = ModelOracleSem.createOracleSemModel(oracle, szModelName,
szUser, szNetworkName);

    model.getGraph().add(Triple.create(
          NodeFactory.createURI("http://example.com/John"),
          NodeFactory.createURI("http://example.com/fatherOf"),
          NodeFactory.createURI("http://example.com/Mary")));

    Query query = QueryFactory.create(
        "select ?f ?k WHERE {?f <http://example.com/fatherOf> ?k .}");
    QueryExecution qexec = QueryExecutionFactory.create(query, model);
    ResultSet results = qexec.execSelect();
    ResultSetFormatter.out(System.out, results, query);

    model.close();
    oracle.dispose();
  }
}
```

The following are the commands to compile and run the preceding code along with the expected output of the `java` command.

```
javac -classpath ../jar/'*' Test_privnet.java
java -classpath ./:../jar/'*'  Test_privnet jdbc:oracle:thin:@localhost:1521:orcl scott
<password-for-scott> M1 NET1

---------------------------------------------------
| f                         | k                     |
===================================================
```

```
| <http://example.com/John> | <http://example.com/Mary> |
---------------------------------------------------------
```

7.16.2 Load OWL Ontology and Perform OWLPrime Inference

The following example loads an OWL ontology and performs OWLPrime inference. Note that the OWL ontology is in RDF/XML format, and after it is loaded into Oracle it will be serialized out in N-TRIPLE form. The example also queries for the number of asserted and inferred triples.

The ontology in this example can be retrieved from `http://swat.cse.lehigh.edu/onto/univ-bench.owl`, and it describes roles, resources, and relationships in a university environment.

Example 7-18 Load OWL Ontology and Perform OWLPrime inference

```
import java.io.*;
import org.apache.jena.query.*;
import org.apache.jena.rdf.model.Model;
import org.apache.jena.util.FileManager;
import oracle.spatial.rdf.client.jena.*;

public class Test6_privnet
{
  public static void main(String[] args) throws Exception
  {
    String szJdbcURL = args[0];
    String szUser    = args[1];
    String szPasswd  = args[2];

    String szModelName = args[3];
    String szNetworkName = args[4];

    Oracle oracle = new Oracle(szJdbcURL, szUser, szPasswd);

    Model model = ModelOracleSem.createOracleSemModel(oracle, szModelName, szUser, 
szNetworkName);

    // load UNIV ontology
    InputStream in = FileManager.get().open("./univ-bench.owl" );

    model.read(in, null);

    OutputStream os = new FileOutputStream("./univ-bench.nt");
    model.write(os, "N-TRIPLE");
    os.close();

    String queryString =
      " SELECT ?subject ?prop ?object WHERE { ?subject ?prop ?object } ";

    Query query = QueryFactory.create(queryString) ;
    QueryExecution qexec = QueryExecutionFactory.create(query, model) ;

    try {
      int iTriplesCount = 0;
      ResultSet results = qexec.execSelect() ;
      for ( ; results.hasNext() ; ) {
        QuerySolution soln = results.nextSolution() ;
        iTriplesCount++;
      }
      System.out.println("Asserted  triples count: " + iTriplesCount);
    }
```

```
    finally {
      qexec.close() ;
    }

    Attachment attachment = Attachment.createInstance(
        new String[] {}, "OWLPRIME",
        InferenceMaintenanceMode.NO_UPDATE,
        QueryOptions.DEFAULT);

    GraphOracleSem graph = new GraphOracleSem(oracle, szModelName, attachment, szUser, 
szNetworkName);
    graph.analyze();
    graph.performInference();


    query = QueryFactory.create(queryString) ;
    qexec = QueryExecutionFactory.create(query,new ModelOracleSem(graph)) ;

    try {
      int iTriplesCount = 0;
      ResultSet results = qexec.execSelect() ;
      for ( ; results.hasNext() ; ) {
        QuerySolution soln = results.nextSolution() ;
        iTriplesCount++;
      }
      System.out.println("Asserted + Infered triples count: " + iTriplesCount);
    }
    finally {
      qexec.close() ;
    }

    OracleUtils.dropSemanticModel(oracle, szModelName, szUser, szNetworkName);

    model.close();
    oracle.dispose();
  }
}
```

The following are the commands to compile and run the preceding code along with the expected output of the `java` command.

```
javac -classpath ../jar/'*' Test6_privnet.java
java -classpath ./:../jar/'*'  Test6_privnet jdbc:oracle:thin:@localhost:1521:orcl scott 
<password-for-scott> M1 NET1
Asserted  triples count: 293
Asserted + Infered triples count: 340
```

Note that this output reflects an older version of the LUBM ontology. The latest version of the ontology has more triples.

7.16.3 Bulk Load OWL Ontology and Perform OWLPrime Inference

The following example loads the same OWL ontology as in Example 7-18, but stored in a local file using Bulk Loader. Ontologies can also be loaded using an incremental and batch loader; these two methods are also listed in the example for completeness.

Example 7-19 Bulk Load OWL Ontology and Perform OWLPrime inference

```
import java.io.*;
import org.apache.jena.query.*;
import org.apache.jena.graph.*;
```

```
import org.apache.jena.rdf.model.*;
import org.apache.jena.util.*;
import oracle.spatial.rdf.client.jena.*;

public class Test7_privnet
{
  public static void main(String[] args) throws Exception
  {
    String szJdbcURL = args[0];
    String szUser    = args[1];
    String szPasswd  = args[2];

    String szModelName = args[3];
    String szNetworkName = args[4];

    // in memory Jena Model
    Model model = ModelFactory.createDefaultModel();
    InputStream is = FileManager.get().open("./univ-bench.owl");
    model.read(is, "", "RDF/XML");
    is.close();

    Oracle oracle = new Oracle(szJdbcURL, szUser, szPasswd);

    ModelOracleSem modelDest = ModelOracleSem.createOracleSemModel(oracle, szModelName,
szUser, szNetworkName);

    GraphOracleSem g = modelDest.getGraph();
    g.dropApplicationTableIndex();

    int method = 2; // try bulk loader
    String tbs = "SYSAUX"; // can be customized
    if (method == 0) {
      System.out.println("start incremental");
      modelDest.add(model);
      System.out.println("end size " + modelDest.size());
    }
    else if (method == 1) {
      System.out.println("start batch load");
      g.getBulkUpdateHandler().addInBatch(
          GraphUtil.findAll(model.getGraph()), tbs);
      System.out.println("end size " + modelDest.size());
    }
    else {
      System.out.println("start bulk load");
      g.getBulkUpdateHandler().addInBulk(
          GraphUtil.findAll(model.getGraph()), tbs);
      System.out.println("end size " + modelDest.size());
    }
    g.rebuildApplicationTableIndex();


    long lCount = g.getCount(Triple.ANY);
    System.out.println("Asserted  triples count: " + lCount);

    model.close();
    oracle.dispose();
  }
}
```

The following are the commands to compile and run the preceding code along with the expected output of the `java` command.

```
javac -classpath ../jar/'*' Test7_privnet.java
java -classpath ./:../jar/'*'  Test7_privnet jdbc:oracle:thin:@localhost:1521:orcl scott
<password-for-scott> M1 NET1
start bulk load
end size 293
Asserted  triples count: 293
```

Note that this output reflects an older version of the LUBM ontology. The latest version of the ontology has more triples.

7.16.4 SPARQL OPTIONAL Query

The following example shows a SPARQL OPTIONAL query. It inserts triples that postulate the following:

- John is a parent of Mary.
- John is a parent of Jack.
- Mary is a parent of Jill.

It then finds parent-child relationships, optionally including any grandchild (gkid) relationships.

Example 7-20   SPARQL OPTIONAL Query

```
import java.io.*;
import org.apache.jena.query.*;
import org.apache.jena.rdf.model.Model;
import org.apache.jena.util.FileManager;
import oracle.spatial.rdf.client.jena.*;
import org.apache.jena.graph.*;

public class Test8_privnet
{
  public static void main(String[] args) throws Exception
  {
    String szJdbcURL = args[0];
    String szUser    = args[1];
    String szPasswd  = args[2];

    String szModelName = args[3];
    String networkName = args[4];

    Oracle oracle = new Oracle(szJdbcURL, szUser, szPasswd);

    ModelOracleSem model = ModelOracleSem.createOracleSemModel(oracle, szModelName,
szUser, networkName);
    GraphOracleSem g = model.getGraph();

    g.add(Triple.create(
          NodeFactory.createURI("u:John"), NodeFactory.createURI("u:parentOf"),
NodeFactory.createURI("u:Mary")));
    g.add(Triple.create(
          NodeFactory.createURI("u:John"), NodeFactory.createURI("u:parentOf"),
NodeFactory.createURI("u:Jack")));
    g.add(Triple.create(
          NodeFactory.createURI("u:Mary"), NodeFactory.createURI("u:parentOf"),
NodeFactory.createURI("u:Jill")));

    String queryString =
      " SELECT ?s ?o ?gkid WHERE { ?s <u:parentOf> ?o . OPTIONAL {?o <u:parentOf> ?
gkid }} ";
```

```
    Query query = QueryFactory.create(queryString) ;
    QueryExecution qexec = QueryExecutionFactory.create(query, model) ;

    try {
      ResultSet results = qexec.execSelect() ;
      ResultSetFormatter.out(System.out, results, query);
    }
    finally {
      qexec.close() ;
    }

    OracleUtils.dropSemanticModel(oracle, szModelName, szUser, networkName);

    model.close();
    oracle.dispose();
  }
}
```

The following are the commands to compile and run the preceding code along with the expected output of the `java` command.

```
javac -classpath ../jar/'*' Test8_privnet.java
java -classpath ./:../jar/'*'  Test8_privnet jdbc:oracle:thin:@localhost:1521:orcl scott
<password-for-scott> M1 NET1
----------------------------------
| s        | o        | gkid     |
==================================
| <u:John> | <u:Mary> | <u:Jill> |
| <u:Mary> | <u:Jill> |          |
| <u:John> | <u:Jack> |          |
----------------------------------
```

7.16.5 SPARQL Query with LIMIT and OFFSET

The following example shows a SPARQL query with LIMIT and OFFSET. It inserts triples that postulate the following:

- John is a parent of Mary.
- John is a parent of Jack.
- Mary is a parent of Jill.

It then finds one parent-child relationship (LIMIT 1), skipping the first two parent-child relationships encountered (OFFSET 2), and optionally includes any grandchild (gkid) relationships for the one found.

Example 7-21 SPARQL Query with LIMIT and OFFSET

```
import java.io.*;
import org.apache.jena.query.*;
import org.apache.jena.rdf.model.Model;
import org.apache.jena.util.FileManager;
import oracle.spatial.rdf.client.jena.*;
import org.apache.jena.graph.*;

public class Test9_privnet
{
  public static void main(String[] args) throws Exception
  {
    String szJdbcURL = args[0];
    String szUser    = args[1];
    String szPasswd  = args[2];
```

```
    String szModelName = args[3];
    String szNetworkName = args[4];

    Oracle oracle = new Oracle(szJdbcURL, szUser, szPasswd);

    ModelOracleSem model = ModelOracleSem.createOracleSemModel(oracle, szModelName,
szUser, szNetworkName);
    GraphOracleSem g = model.getGraph();

    g.add(Triple.create(
          NodeFactory.createURI("u:John"), NodeFactory.createURI("u:parentOf"),
NodeFactory.createURI("u:Mary")));
    g.add(Triple.create(
          NodeFactory.createURI("u:John"), NodeFactory.createURI("u:parentOf"),
NodeFactory.createURI("u:Jack")));
    g.add(Triple.create(
          NodeFactory.createURI("u:Mary"), NodeFactory.createURI("u:parentOf"),
NodeFactory.createURI("u:Jill")));

    String queryString =
      " SELECT ?s ?o ?gkid WHERE { ?s <u:parentOf> ?o . OPTIONAL {?o <u:parentOf> ?
gkid }} "
      + " LIMIT 1 OFFSET 2";

    Query query = QueryFactory.create(queryString) ;
    QueryExecution qexec = QueryExecutionFactory.create(query, model) ;

    try {
      ResultSet results = qexec.execSelect() ;
      ResultSetFormatter.out(System.out, results, query);
    }
    finally {
      qexec.close() ;
    }

    OracleUtils.dropSemanticModel(oracle, szModelName, szUser, szNetworkName);

    model.close();
    oracle.dispose();
  }
}
```

The following are the commands to compile and run the preceding code along with the expected output of the `java` command.

```
javac -classpath ../jar/'*' Test9_privnet.java
java -classpath ./:../jar/'*'  Test9_privnet jdbc:oracle:thin:@localhost:1521:orcl scott
<password-for-scott> M1 NET1
------------------------------
| s        | o        | gkid |
==============================
| <u:John> | <u:Jack> |      |
------------------------------
```

7.16.6 SPARQL Query with TIMEOUT and DOP

The following example shows the SPARQL query from Example 7-21 with additional features, including a timeout setting (TIMEOUT=1, in seconds) and parallel execution setting (DOP=4).

Example 7-22   SPARQL Query with TIMEOUT and DOP

```
import org.apache.jena.query.*;
import oracle.spatial.rdf.client.jena.*;
import org.apache.jena.graph.*;

public class Test10_privnet
{
  public static void main(String[] args) throws Exception
  {
    String szJdbcURL = args[0];
    String szUser    = args[1];
    String szPasswd  = args[2];

    String szModelName = args[3];
    String szNetworkName = args[4];

    Oracle oracle = new Oracle(szJdbcURL, szUser, szPasswd);

    ModelOracleSem model = ModelOracleSem.createOracleSemModel(oracle, szModelName,
szUser, szNetworkName);
    GraphOracleSem g = model.getGraph();

    g.add(Triple.create(
          NodeFactory.createURI("u:John"), NodeFactory.createURI("u:parentOf"),
NodeFactory.createURI("u:Mary")));
    g.add(Triple.create(
          NodeFactory.createURI("u:John"), NodeFactory.createURI("u:parentOf"),
NodeFactory.createURI("u:Jack")));
    g.add(Triple.create(
          NodeFactory.createURI("u:Mary"), NodeFactory.createURI("u:parentOf"),
NodeFactory.createURI("u:Jill")));

    String queryString =
        " PREFIX ORACLE_SEM_FS_NS: <http://oracle.com/semtech#dop=4,timeout=1> "
      + " SELECT ?s ?o ?gkid WHERE { ?s <u:parentOf> ?o . OPTIONAL {?o <u:parentOf> ?
gkid }} "
      + " LIMIT 1 OFFSET 2";

    Query query = QueryFactory.create(queryString) ;
    QueryExecution qexec = QueryExecutionFactory.create(query, model) ;

    try {
      ResultSet results = qexec.execSelect() ;
      ResultSetFormatter.out(System.out, results, query);
    }
    finally {
      qexec.close() ;
    }

    OracleUtils.dropSemanticModel(oracle, szModelName, szUser, szNetworkName);

    model.close();
    oracle.dispose();
  }
}
```

The following are the commands to compile and run the preceding code along with the expected output of the `java` command.

```
javac -classpath ../jar/'*' Test10_privnet.java
java -classpath ./:../jar/'*'  Test10_privnet jdbc:oracle:thin:@localhost:1521:orcl
```

```
scott <password-for-scott> M1 NET1
------------------------------
| s         | o         | gkid |
==============================
| <u:John> | <u:Jack> |      |
------------------------------
```

7.16.7 Query Involving Named Graphs

The following example shows a query involving named graphs. It involves a default graph that has information about named graph URIs and their publishers. The query finds graph names, their publishers, and within each named graph finds the mailbox value using the `foaf:mbox` predicate.

Example 7-23 Named Graph Based Query

```
import org.apache.jena.query.*;
import oracle.spatial.rdf.client.jena.*;
import org.apache.jena.graph.*;

public class Test11_privnet
{
  public static void main(String[] args) throws Exception
  {
    String szJdbcURL = args[0];
    String szUser    = args[1];
    String szPasswd  = args[2];

    String szModelName = args[3];
    String szNetworkName = args[4];

    Oracle oracle = new Oracle(szJdbcURL, szUser, szPasswd);

    Dataset ds = DatasetFactory.create();

    ModelOracleSem model = ModelOracleSem.createOracleSemModel(oracle, szModelName,
szUser, szNetworkName);
    model.getGraph().add(Triple.create(NodeFactory.createURI("http://example.org/bob"),
          NodeFactory.createURI("http://purl.org/dc/elements/1.1/publisher"),
          NodeFactory.createLiteral("Bob Hacker")));
    model.getGraph().add(Triple.create(NodeFactory.createURI("http://example.org/alice"),
          NodeFactory.createURI("http://purl.org/dc/elements/1.1/publisher"),
          NodeFactory.createLiteral("alice Hacker")));


    ModelOracleSem model1 = ModelOracleSem.createOracleSemModel(oracle, szModelName+"1",
szUser, szNetworkName);

    model1.getGraph().add(Triple.create(NodeFactory.createURI("urn:bob"),
                                        NodeFactory.createURI("http://xmlns.com/foaf/0.1/
name"),
                                        NodeFactory.createLiteral("Bob")
          ));
    model1.getGraph().add(Triple.create(NodeFactory.createURI("urn:bob"),
                                        NodeFactory.createURI("http://xmlns.com/foaf/0.1/
mbox"),
                                        NodeFactory.createURI("mailto:bob@example")
          ));

    ModelOracleSem model2 = ModelOracleSem.createOracleSemModel(oracle, szModelName+"2",
szUser, szNetworkName);
    model2.getGraph().add(Triple.create(NodeFactory.createURI("urn:alice"),
```

```
                                        NodeFactory.createURI("http://xmlns.com/foaf/0.1/
name"),
                                        NodeFactory.createLiteral("Alice")
          ));
    model2.getGraph().add(Triple.create(NodeFactory.createURI("urn:alice"),
                                        NodeFactory.createURI("http://xmlns.com/foaf/0.1/
mbox"),
                                        NodeFactory.createURI("mailto:alice@example")
          ));

    ds.setDefaultModel(model);
    ds.addNamedModel("http://example.org/bob",model1);
    ds.addNamedModel("http://example.org/alice",model2);

    String queryString =
      " PREFIX foaf: <http://xmlns.com/foaf/0.1/> "
    + " PREFIX dc: <http://purl.org/dc/elements/1.1/> "
    + " SELECT ?who ?graph ?mbox "
    + " FROM NAMED <http://example.org/alice> "
    + " FROM NAMED <http://example.org/bob> "
    + " WHERE "
    + " { "
    + "    ?graph dc:publisher ?who . "
    + "    GRAPH ?graph { ?x foaf:mbox ?mbox } "
    + " } ";

    Query query = QueryFactory.create(queryString) ;
    QueryExecution qexec = QueryExecutionFactory.create(query, ds) ;

    ResultSet results = qexec.execSelect() ;
    ResultSetFormatter.out(System.out, results, query);

    qexec.close();
    model.close();
    model1.close();
    model2.close();

    OracleUtils.dropSemanticModel(oracle, szModelName, szUser, szNetworkName);
    OracleUtils.dropSemanticModel(oracle, szModelName + "1", szUser, szNetworkName);
    OracleUtils.dropSemanticModel(oracle, szModelName + "2", szUser, szNetworkName);
    oracle.dispose();
  }
}
```

The following are the commands to compile and run the preceding code along with the expected output of the `java` command.

```
javac -classpath ./:./jena-2.6.4.jar:./sdordfclient.jar:./ojdbc6.jar:./slf4j-
api-1.5.8.jar:./slf4j-log4j12-1.5.8.jar:./arq-2.8.8.jar:./xercesImpl-2.7.1.jar
Test11_privnet.java
java -classpath ./:../jar/'*'  Test11_privnet jdbc:oracle:thin:@localhost:1521:orcl
scott <password-for-scott> M1 NET1
------------------------------------------------------------------------
| who            | graph                        | mbox                   |
========================================================================
| "alice Hacker" | <http://example.org/alice> | <mailto:alice@example> |
| "Bob Hacker"   | <http://example.org/bob>   | <mailto:bob@example>   |
------------------------------------------------------------------------
```

7.16.8 SPARQL ASK Query

The following example shows a SPARQL ASK query. It inserts a triple that postulates that John is a parent of Mary. It then finds whether John is a parent of Mary.

Example 7-24 SPARQL ASK Query

```
import org.apache.jena.query.*;
import oracle.spatial.rdf.client.jena.*;
import org.apache.jena.graph.*;

public class Test12_privnet
{
  public static void main(String[] args) throws Exception
  {
    String szJdbcURL = args[0];
    String szUser    = args[1];
    String szPasswd  = args[2];

    String szModelName = args[3];
    String szNetworkName = args[4];

    Oracle oracle = new Oracle(szJdbcURL, szUser, szPasswd);

    ModelOracleSem model = ModelOracleSem.createOracleSemModel(oracle, szModelName, 
szUser, szNetworkName);
    GraphOracleSem g = model.getGraph();

    g.add(Triple.create(
          NodeFactory.createURI("u:John"), NodeFactory.createURI("u:parentOf"), 
NodeFactory.createURI("u:Mary")));
    String queryString = " ASK { <u:John> <u:parentOf> <u:Mary> } ";

    Query query = QueryFactory.create(queryString) ;
    QueryExecution qexec = QueryExecutionFactory.create(query, model) ;
    boolean b = qexec.execAsk();
    System.out.println("ask result = " + ((b)?"TRUE":"FALSE"));
    qexec.close() ;

    OracleUtils.dropSemanticModel(oracle, szModelName, szUser, szNetworkName);

    model.close();
    oracle.dispose();
  }
}
```

The following are the commands to compile and run the preceding code along with the expected output of the `java` command.

```
javac -classpath ../jar/'*' Test12_privnet.java
java -classpath ./:../jar/'*'  Test12_privnet jdbc:oracle:thin:@localhost:1521:orcl 
scott <password-for-scott> M1 NET1
ask result = TRUE
```

7.16.9 SPARQL DESCRIBE Query

The following example shows a SPARQL DESCRIBE query. It inserts triples that postulate the following:

- John is a parent of Mary.

- John is a parent of Jack.
- Amy is a parent of Jack.

It then finds all relationships that involve any parents of Jack.

Example 7-25   SPARQL DESCRIBE Query

The following are the commands to compile and run the preceding code along with the expected output of the `java` command.

```
import org.apache.jena.query.*;
import org.apache.jena.rdf.model.Model;
import oracle.spatial.rdf.client.jena.*;
import org.apache.jena.graph.*;

public class Test13_privnet
{
  public static void main(String[] args) throws Exception
  {
    String szJdbcURL = args[0];
    String szUser    = args[1];
    String szPasswd  = args[2];

    String szModelName = args[3];
    String szNetworkName = args[4];

    Oracle oracle = new Oracle(szJdbcURL, szUser, szPasswd);

    ModelOracleSem model = ModelOracleSem.createOracleSemModel(oracle, szModelName,
szUser, szNetworkName);
    GraphOracleSem g = model.getGraph();

    g.add(Triple.create(
          NodeFactory.createURI("u:John"), NodeFactory.createURI("u:parentOf"),
NodeFactory.createURI("u:Mary")));
    g.add(Triple.create(
          NodeFactory.createURI("u:John"), NodeFactory.createURI("u:parentOf"),
NodeFactory.createURI("u:Jack")));
    g.add(Triple.create(
          NodeFactory.createURI("u:Amy"), NodeFactory.createURI("u:parentOf"),
NodeFactory.createURI("u:Jack")));
    String queryString = " DESCRIBE ?x WHERE {?x <u:parentOf> <u:Jack>}";

    Query query = QueryFactory.create(queryString) ;
    QueryExecution qexec = QueryExecutionFactory.create(query, model) ;
    Model m = qexec.execDescribe();
    System.out.println("describe result = " + m.toString());

    qexec.close() ;
    OracleUtils.dropSemanticModel(oracle, szModelName, szUser, szNetworkName);
    model.close();
    oracle.dispose();
  }
}
```

The following are the commands to compile and run the preceding code along with the expected output of the `java` command.

```
javac -classpath ../jar/'*' Test13_privnet.java
java -classpath ./:../jar/'*'  Test13_privnet jdbc:oracle:thin:@localhost:1521:orcl
scott <password-for-scott> M1 NET1
describe result = <ModelCom   {u:Amy @u:parentOf u:Jack;
```

```
     u:John @u:parentOf u:Jack; u:John @u:parentOf u:Mary} |  [u:Amy, u:parentOf,
u:Jack] [u:John, u:parentOf,
       u:Jack] [u:John, u:parentOf, u:Mary]>
```

7.16.10 SPARQL CONSTRUCT Query

The following example shows a SPARQL CONSTRUCT query. It inserts triples that postulate the following:

- John is a parent of Mary.
- John is a parent of Jack.
- Amy is a parent of Jack.
- Each parent loves their children.

It then constructs an RDF graph with information about who loves whom.

Example 7-26   SPARQL CONSTRUCT Query

The following are the commands to compile and run the preceding code along with the expected output of the `java` command.

```
import org.apache.jena.query.*;
import org.apache.jena.rdf.model.Model;
import oracle.spatial.rdf.client.jena.*;
import org.apache.jena.graph.*;

public class Test14_privnet
{
  public static void main(String[] args) throws Exception
  {
    String szJdbcURL = args[0];
    String szUser    = args[1];
    String szPasswd  = args[2];

    String szModelName = args[3];
    String szNetworkName = args[4];

    Oracle oracle = new Oracle(szJdbcURL, szUser, szPasswd);

    ModelOracleSem model = ModelOracleSem.createOracleSemModel(oracle, szModelName,
szUser, szNetworkName);
    GraphOracleSem g = model.getGraph();

    g.add(Triple.create(
          NodeFactory.createURI("u:John"), NodeFactory.createURI("u:parentOf"),
NodeFactory.createURI("u:Mary")));
    g.add(Triple.create(
          NodeFactory.createURI("u:John"), NodeFactory.createURI("u:parentOf"),
NodeFactory.createURI("u:Jack")));
    g.add(Triple.create(
          NodeFactory.createURI("u:Amy"), NodeFactory.createURI("u:parentOf"),
NodeFactory.createURI("u:Jack")));
    String queryString = " CONSTRUCT { ?s <u:loves> ?o } WHERE {?s <u:parentOf> ?o}";

    Query query = QueryFactory.create(queryString) ;
    QueryExecution qexec = QueryExecutionFactory.create(query, model) ;
    Model m = qexec.execConstruct();
    System.out.println("Construct result = " + m.toString());

    qexec.close() ;
```

```
    OracleUtils.dropSemanticModel(oracle, szModelName, szUser, szNetworkName);
    model.close();
    oracle.dispose();
  }
}
```

The following are the commands to compile and run the preceding code along with the expected output of the `java` command.

```
javac -classpath ../jar/'*' Test14_privnet.java
java -classpath ./:../jar/'*'  Test14_privnet jdbc:oracle:thin:@localhost:1521:orcl
scott <password-for-scott> M1 NET1
Construct result = <ModelCom   {u:Amy @u:loves u:Jack;
  u:John @u:loves u:Jack; u:John @u:loves u:Mary} |  [u:Amy, u:loves, u:Jack] [u:John,
u:loves,
    u:Jack] [u:John, u:loves, u:Mary]>
```

7.16.11 Query Multiple Models and Specify "Allow Duplicates"

The following example queries multiple models and uses the "allow duplicates" option. It inserts triples that postulate the following:

- John is a parent of Jack (in Model 1)
- Mary is a parent of Jack (in Model 2)
- Each parent loves their children.

It then finds out who loves whom. It searches both models and allows for the possibility of duplicate triples in the models (although there are no duplicates in this example).

Example 7-27 Query Multiple Models and Specify "Allow Duplicates"

The following are the commands to compile and run the preceding code along with the expected output of the `java` command.

```
import org.apache.jena.query.*;
import org.apache.jena.rdf.model.Model;
import oracle.spatial.rdf.client.jena.*;
import org.apache.jena.graph.*;

public class Test15_privnet
{
  public static void main(String[] args) throws Exception
  {
    String szJdbcURL = args[0];
    String szUser    = args[1];
    String szPasswd  = args[2];

    String szModelName1 = args[3];
    String szModelName2 = args[4];
    String szNetworkName = args[5];

    Oracle oracle = new Oracle(szJdbcURL, szUser, szPasswd);

    ModelOracleSem model1 = ModelOracleSem.createOracleSemModel(oracle, szModelName1,
szUser, szNetworkName);
    model1.getGraph().add(Triple.create(
          NodeFactory.createURI("u:John"), NodeFactory.createURI("u:parentOf"),
NodeFactory.createURI("u:Jack")));
    model1.close();

    ModelOracleSem model2 = ModelOracleSem.createOracleSemModel(oracle, szModelName2,
```

```
szUser, szNetworkName);
    model2.getGraph().add(Triple.create(
          NodeFactory.createURI("u:Mary"), NodeFactory.createURI("u:parentOf"),
NodeFactory.createURI("u:Jack")));
    model2.close();


    String[] modelNamesList = {szModelName2};
    String[] rulebasesList  = {};

    Attachment attachment = Attachment.createInstance(modelNamesList, rulebasesList,
              InferenceMaintenanceMode.NO_UPDATE,
QueryOptions.ALLOW_QUERY_VALID_AND_DUP);

    GraphOracleSem graph = new GraphOracleSem(oracle, szModelName1, attachment, szUser,
szNetworkName);
    ModelOracleSem model = new ModelOracleSem(graph);

    String queryString = " CONSTRUCT { ?s <u:loves> ?o } WHERE {?s <u:parentOf> ?o}";

    Query query = QueryFactory.create(queryString) ;
    QueryExecution qexec = QueryExecutionFactory.create(query, model) ;
    Model m = qexec.execConstruct();
    System.out.println("Construct result = " + m.toString());

    qexec.close() ;
    model.close();

    OracleUtils.dropSemanticModel(oracle, szModelName1, szUser, szNetworkName);
    OracleUtils.dropSemanticModel(oracle, szModelName2, szUser, szNetworkName);
    oracle.dispose();
  }
}
```

The following are the commands to compile and run the preceding code along with the expected output of the `java` command.

```
javac -classpath ../jar/'*' Test15_privnet.java
java -classpath ./:../jar/'*'  Test15_privnet jdbc:oracle:thin:@localhost:1521:orcl
scott <password-for-scott> M1 M2 NET1
Construct result = <ModelCom   {u:Mary @u:loves u:Jack; u:John @u:loves u:Jack} |
[u:Mary, u:loves, u:Jack] [u:John, u:loves, u:Jack]>
```

7.16.12 SPARQL Update

The following example inserts two triples into a model.

Example 7-28   SPARQL Update

```
import org.apache.jena.util.iterator.*;
import oracle.spatial.rdf.client.jena.*;
import org.apache.jena.graph.*;
import org.apache.jena.update.*;

public class Test16_privnet
{
  public static void main(String[] args) throws Exception
  {
    String szJdbcURL = args[0];
    String szUser    = args[1];
    String szPasswd  = args[2];
```

```
    String szModelName = args[3];
    String szNetworkName = args[4];

    Oracle oracle = new Oracle(szJdbcURL, szUser, szPasswd);

    ModelOracleSem model = ModelOracleSem.createOracleSemModel(oracle, szModelName,
szUser, szNetworkName);
    GraphOracleSem g = model.getGraph();
    String insertString =
      " PREFIX dc: <http://purl.org/dc/elements/1.1/> "         +
      " INSERT DATA "                                           +
      " { <http://example/book3> dc:title    \"A new book\" ; " +
      "                         dc:creator  \"A.N.Other\" . "   +
      " }   ";

    UpdateAction.parseExecute(insertString,  model);

    ExtendedIterator<Triple> ei = GraphUtil.findAll(g);
    while (ei.hasNext()) {
      System.out.println("Triple " + ei.next().toString());
    }
    OracleUtils.dropSemanticModel(oracle, szModelName, szUser, szNetworkName);
    model.close();
    oracle.dispose();
  }
}
```

The following are the commands to compile and run the preceding code along with the expected output of the `java` command.

```
javac -classpath ../jar/'*' Test16_privnet.java
java -classpath ./:../jar/'*'  Test16_privnet jdbc:oracle:thin:@localhost:1521:orcl
scott <password-for-scott> M1 NET1
Triple http://example/book3 @dc:title "A new book"
Triple http://example/book3 @dc:creator "A.N.Other"
```

7.16.13 SPARQL Query with ARQ Built-In Functions

The following example inserts data about two books, and it displays the book titles in all uppercase characters and the length of each title string.

Example 7-29 SPARQL Query with ARQ Built-In Functions

```
import org.apache.jena.query.*;
import oracle.spatial.rdf.client.jena.*;
import org.apache.jena.update.*;

public class Test17_privnet
{
  public static void main(String[] args) throws Exception
  {
    String szJdbcURL = args[0];
    String szUser    = args[1];
    String szPasswd  = args[2];

    String szModelName = args[3];
    String szNetworkName = args[4];

    Oracle oracle = new Oracle(szJdbcURL, szUser, szPasswd);

    ModelOracleSem model = ModelOracleSem.createOracleSemModel(oracle, szModelName,
szUser, szNetworkName);
```

```
    GraphOracleSem g = model.getGraph();
    String insertString =
      " PREFIX dc: <http://purl.org/dc/elements/1.1/> "         +
      " INSERT DATA "                                            +
      " { <http://example/book3> dc:title    \"A new book\" ; " +
      "                        dc:creator  \"A.N.Other\" . "    +
      "   <http://example/book4> dc:title    \"Semantic Web Rocks\" ; " +
      "                        dc:creator  \"TB\" . "    +
      " }   ";

    UpdateAction.parseExecute(insertString,  model);

    String queryString = "PREFIX  dc:   <http://purl.org/dc/elements/1.1/> " +
      " PREFIX  fn: <http://www.w3.org/2005/xpath-functions#> " +
      " SELECT ?subject (fn:upper-case(?object) as ?object1) (fn:string-length(?object)
as ?strlen) " +
      "WHERE { ?subject dc:title ?object } "
      ;

    Query query = QueryFactory.create(queryString, Syntax.syntaxARQ);
    QueryExecution qexec = QueryExecutionFactory.create(query, model);
    ResultSet results = qexec.execSelect();

    ResultSetFormatter.out(System.out, results, query);

    model.close();
    OracleUtils.dropSemanticModel(oracle, szModelName, szUser, szNetworkName);
    oracle.dispose();
  }
}
```

The following are the commands to compile and run the preceding code along with the expected output of the `java` command.

```
javac -classpath ../jar/'*' Test17_privnet.java
java -classpath ./:../jar/'*'  Test17_privnet jdbc:oracle:thin:@localhost:1521:orcl
scott <password-for-scott> M1 NET1
----------------------------------------------------------
| subject                | object1              | strlen |
==========================================================
| <http://example/book3> | "A NEW BOOK"         | 10     |
| <http://example/book4> | "SEMANTIC WEB ROCKS" | 18     |
----------------------------------------------------------
```

7.16.14 SELECT Cast Query

The following example "converts" two Fahrenheit temperatures (18.1 and 32.0) to Celsius temperatures.

Example 7-30 SELECT Cast Query

```
import org.apache.jena.query.*;
import oracle.spatial.rdf.client.jena.*;
import org.apache.jena.update.*;

public class Test18_privnet
{
  public static void main(String[] args) throws Exception
  {
    String szJdbcURL = args[0];
    String szUser    = args[1];
    String szPasswd  = args[2];
```

```
    String szModelName = args[3];
    String szNetworkName = args[4];

    Oracle oracle = new Oracle(szJdbcURL, szUser, szPasswd);

    ModelOracleSem model = ModelOracleSem.createOracleSemModel(oracle, szModelName,
szUser, szNetworkName);
    GraphOracleSem g = model.getGraph();
    String insertString =
      " PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> " +
      " INSERT DATA "                                     +
      " { <u:Object1> <u:temp>    \"18.1\"^^xsd:float ; " +
      "               <u:name>    \"Foo... \" . "         +
      "   <u:Object2> <u:temp>    \"32.0\"^^xsd:float ; " +
      "               <u:name>    \"Bar... \" . "         +
      " }   ";

    UpdateAction.parseExecute(insertString,  model);

    String queryString =
      " PREFIX  fn: <http://www.w3.org/2005/xpath-functions#> " +
      " SELECT ?subject ((?temp - 32.0)*5/9 as ?celsius_temp) " +
      "WHERE { ?subject <u:temp> ?temp } "
      ;

    Query query = QueryFactory.create(queryString, Syntax.syntaxARQ);
    QueryExecution qexec = QueryExecutionFactory.create(query, model);
    ResultSet results = qexec.execSelect();

    ResultSetFormatter.out(System.out, results, query);

    model.close();
    OracleUtils.dropSemanticModel(oracle, szModelName, szUser, szNetworkName);
    oracle.dispose();
  }
}
```

The following are the commands to compile and run the preceding code along with the expected output of the `java` command.

```
javac -classpath ../jar/'*' Test18_privnet.java
java -classpath ./:../jar/'*'  Test18_privnet jdbc:oracle:thin:@localhost:1521:orcl
scott <password-for-scott> M1 NET1
------------------------------------------------------------------------
| subject     | celsius_temp                                           |
========================================================================
| <u:Object1> | "-7.7222223"^^<http://www.w3.org/2001/XMLSchema#float> |
| <u:Object2> | "0.0"^^<http://www.w3.org/2001/XMLSchema#float>        |
------------------------------------------------------------------------
```

7.16.15 Instantiate Oracle Database Using OracleConnection

The following example shows a different way to instantiate an Oracle object using a given `OracleConnection` object. (In a J2EE Web application, users can normally get an `OracleConnection` object from a J2EE data source.)

Example 7-31 Instantiate Oracle Database Using OracleConnection

The following are the commands to compile and run the preceding code along with the expected output of the `java` command.

```
import org.apache.jena.query.*;
import org.apache.jena.graph.*;
import oracle.spatial.rdf.client.jena.*;
import oracle.jdbc.pool.*;
import oracle.jdbc.*;

public class Test19_privnet
{
  public static void main(String[] args) throws Exception
  {
    String szJdbcURL = args[0];
    String szUser    = args[1];
    String szPasswd  = args[2];

    String szModelName = args[3];
    String szNetworkName = args[4];

    OracleDataSource ds = new OracleDataSource();
    ds.setURL(szJdbcURL);
    ds.setUser(szUser);
    ds.setPassword(szPasswd);

    OracleConnection conn = (OracleConnection) ds.getConnection();
    Oracle oracle = new Oracle(conn);

    ModelOracleSem model = ModelOracleSem.createOracleSemModel(oracle, szModelName,
szUser, szNetworkName);
    GraphOracleSem g = model.getGraph();

    g.add(Triple.create(
          NodeFactory.createURI("u:John"), NodeFactory.createURI("u:parentOf"),
NodeFactory.createURI("u:Mary")));
    g.add(Triple.create(
          NodeFactory.createURI("u:John"), NodeFactory.createURI("u:parentOf"),
NodeFactory.createURI("u:Jack")));
    g.add(Triple.create(
          NodeFactory.createURI("u:Mary"), NodeFactory.createURI("u:parentOf"),
NodeFactory.createURI("u:Jill")));

    String queryString =
       " SELECT ?s ?o  WHERE { ?s <u:parentOf> ?o .} ";

    Query query = QueryFactory.create(queryString) ;
    QueryExecution qexec = QueryExecutionFactory.create(query, model) ;

    ResultSet results = qexec.execSelect() ;
    ResultSetFormatter.out(System.out, results, query);
    qexec.close() ;

    OracleUtils.dropSemanticModel(oracle, szModelName, szUser, szNetworkName);

    model.close();
    oracle.dispose();
  }
}
```

The following are the commands to compile and run the preceding code along with the expected output of the `java` command.

```
javac -classpath ../jar/'*' Test19_privnet.java
java -classpath ./:../jar/'*'  Test19_privnet jdbc:oracle:thin:@localhost:1521:orcl
scott <password-for-scott> M1 NET1
```

```
-----------------------
| s        | o        |
=======================
| <u:John> | <u:Mary> |
| <u:John> | <u:Jack> |
| <u:Mary> | <u:Jill> |
-----------------------
```

7.16.16 Oracle Database Connection Pooling

The following example uses Oracle Database connection pooling.

Example 7-32 Oracle Database Connection Pooling

The following are the commands to compile and run the preceding code along with the expected output of the `java` command.

```
import org.apache.jena.graph.*;
import oracle.spatial.rdf.client.jena.*;

public class Test20_privnet
{
  public static void main(String[] args) throws Exception
  {
    String szJdbcURL = args[0];
    String szUser    = args[1];
    String szPasswd  = args[2];

    String szModelName = args[3];
    String szNetworkName = args[4];

    // test with connection properties (taken from some example)
    java.util.Properties prop = new java.util.Properties();
    prop.setProperty("MinLimit", "2");     // the cache size is 2 at least
    prop.setProperty("MaxLimit", "10");
    prop.setProperty("InitialLimit", "2"); // create 2 connections at startup
    prop.setProperty("InactivityTimeout", "1800");    //  seconds
    prop.setProperty("AbandonedConnectionTimeout", "900");  //  seconds
    prop.setProperty("MaxStatementsLimit", "10");
    prop.setProperty("PropertyCheckInterval", "60"); // seconds

    System.out.println("Creating OraclePool");
    OraclePool op = new OraclePool(szJdbcURL, szUser, szPasswd, prop,
"OracleSemConnPool");
    System.out.println("Done creating OraclePool");

    // grab an Oracle and do something with it
    System.out.println("Getting an Oracle from OraclePool");
    Oracle oracle = op.getOracle();
    System.out.println("Done");
    System.out.println("Is logical connection:" +
        oracle.getConnection().isLogicalConnection());
    GraphOracleSem g = new GraphOracleSem(oracle, szModelName, szUser, szNetworkName);
    g.add(Triple.create(
          NodeFactory.createURI("u:John"), NodeFactory.createURI("u:parentOf"),
NodeFactory.createURI("u:Mary")));
    g.close();
    // return the Oracle back to the pool
    oracle.dispose();

    // grab another Oracle and do something else
    System.out.println("Getting an Oracle from OraclePool");
```

```
    oracle = op.getOracle();
    System.out.println("Done");
    System.out.println("Is logical connection:" +
        oracle.getConnection().isLogicalConnection());
    g = new GraphOracleSem(oracle, szModelName, szUser, szNetworkName);
    g.add(Triple.create(
          NodeFactory.createURI("u:John"), NodeFactory.createURI("u:parentOf"),
NodeFactory.createURI("u:Jack")));
    g.close();

    OracleUtils.dropSemanticModel(oracle, szModelName, szUser, szNetworkName);

    // return the Oracle back to the pool
    oracle.dispose();
  }
}
```

The following are the commands to compile and run the preceding code along with the expected output of the `java` command.

```
javac -classpath ../jar/'*' Test20_privnet.java
java -classpath ./:../jar/'*'  Test20_privnet jdbc:oracle:thin:@localhost:1521:orcl
scott <password-for-scott> M1 NET1
Creating OraclePool
Done creating OraclePool
Getting an Oracle from OraclePool
Done
Is logical connection:true
Getting an Oracle from OraclePool
Done
Is logical connection:true
```

7.17 SPARQL Gateway and RDF Data

SPARQL Gateway is a J2EE web application that is included with the support for Apache Jena. It is designed to make RDF data (RDF/OWL/SKOS) easily available to applications that operate on relational and XML data, including Oracle Business Intelligence Enterprise Edition (OBIEE) 11*g*.

- SPARQL Gateway Features and Benefits Overview
- Installing and Configuring SPARQL Gateway
- Using SPARQL Gateway with RDF Data
- Customizing the Default XSLT File
- Using the SPARQL Gateway Java API
- Using the SPARQL Gateway Graphical Web Interface
- Using SPARQL Gateway as an XML Data Source to OBIEE

7.17.1 SPARQL Gateway Features and Benefits Overview

SPARQL Gateway handles several challenges in exposing RDF data to a non-semantic application:

- RDF syntax, SPARQL query syntax and SPARQL protocol must be understood.
- The SPARQL query response syntax must be understood.

- A transformation must convert a SPARQL query response to something that the application can consume.

To address these challenges, SPARQL Gateway manages SPARQL queries and XSLT operations, executes SPARQL queries against any arbitrary standard-compliant SPARQL endpoints, and performs necessary XSL transformations before passing the response back to applications. Applications can then consume RDF data as if it is coming from an existing data source.

Different triple stores or quad stores often have different capabilities. For example, the SPARQL endpoint supported by Oracle Database, with RDF Graph support for Apache Jena, allows parallel execution, query timeout, dynamic sampling, result cache, and other features, in addition to the core function of parsing and answering a given standard-compliant SPARQL query. However, these features may not be available from another given RDF data store.

With the RDF Graph SPARQL Gateway, you get certain highly desirable capabilities, such as the ability to set a timeout on a long running query and the ability to get partial results from a complex query in a given amount of time. Waiting indefinitely for a query to finish is a challenge for end users, as is an application with a response time constraint. SPARQL Gateway provides both timeout and best effort query functions on top of a SPARQL endpoint. This effectively removes some uncertainty from consuming RDF data through SPARQL query executions. (See Specifying a Timeout Value and Specifying Best Effort Query Execution.)

7.17.2 Installing and Configuring SPARQL Gateway

To install and configure SPARQL Gateway, follow these major steps, which are explained in their own topics:

1. Download the RDF Graph Support for Apache Jena .zip File (if Not Already Done)
2. Deploy SPARQL Gateway in WebLogic Server
3. Modify Proxy Settings_ if Necessary
4. Configure the OracleSGDS Data Source_ if Necessary
5. Add and Configure the SparqlGatewayAdminGroup Group_ if Desired

- Download the RDF Graph Support for Apache Jena .zip File (if Not Already Done)
- Deploy SPARQL Gateway in WebLogic Server
- Modify Proxy Settings, if Necessary
- Configure the OracleSGDS Data Source, if Necessary
- Add and Configure the SparqlGatewayAdminGroup Group, if Desired

7.17.2.1 Download the RDF Graph Support for Apache Jena .zip File (if Not Already Done)

If you have not already done so, download the RDF Graph support for Apache Jena file from the RDF Graph page and unzip it into a temporary directory, as explained in Setting Up the Software Environment.

Note that the SPARQL Gateway Java class implementations are embedded in `sdordfclient.jar` (see Using the SPARQL Gateway Java API).

7.17.2.2 Deploy SPARQL Gateway in WebLogic Server

Deploy SPARQL Gateway in Oracle WebLogic Server, as follows:

1. Go to the autodeploy directory of WebLogic Server, and copy over the prebuilt `sparqlgateway.war` file as follows. (For information about auto-deploying applications in development domains, see: `http://docs.oracle.com/cd/E11035_01/wls100/deployment/autodeploy.html`)

   ```
   cp -rf  /tmp/jena_adapter/sparqlgateway_web_app/sparqlgateway.war  <domain_name>/
   autodeploy/sparqgateway.war
   ```

   In this example, <*domain_name*> is the name of a WebLogic Server domain.

   You can customize the prebuilt application in the following ways:

   - Modify the `WEB-INF/web.xml` file embedded in `sparqlgateway_web_app/sparqlgateway.war` as needed. Be sure to specify appropriate values for the `sparql_gateway_repository_filedir` and `sparql_gateway_repository_url` parameters.
   - Add XSLT files or SPARQL query files to the top-level directory of `sparqlgateway_web_app/sparqlgateway.war`, if necessary.

     The following files are provided by Oracle in that directory: `default.xslt`, `noop.xslt`, and `qb1.sparql`. The `default.xslt` file is intended mainly for transforming SPARQL query responses (XML) to a format acceptable to Oracle.

     (These files are described in Storing SPARQL Queries and XSL Transformations; using SPARQL Gateway with OBIEE is explained in Using SPARQL Gateway as an XML Data Source to OBIEE.)

2. Verify your deployment by using your Web browser to connect to a URL in the following format (assume that the Web application is deployed at port 7001):

   ```
   http://<hostname>:7001/sparqlgateway
   ```

7.17.2.3 Modify Proxy Settings, if Necessary

If your SPARQL Gateway is behind a firewall and you want SPARQL Gateway to communicate with SPARQL endpoints on the Internet as well as those inside the firewall, you probably need to use the following JVM settings:

```
-Dhttp.proxyHost=<your_proxy_host>
-Dhttp.proxyPort=<your_proxy_port>
-Dhttp.nonProxyHosts=127.0.0.1|<hostname_1_for_sparql_endpoint_inside_firewall>|
<hostname_2_for_sparql_endpoint_inside_firewall>|...|
<hostname_n_for_sparql_endpoint_inside_firewall>
```

You can specify these settings in the `startWebLogic.sh` script.

7.17.2.4 Configure the OracleSGDS Data Source, if Necessary

If an Oracle database is used for storage of and access to SPARQL queries and XSL transformations for SPARQL Gateway, then a data source named `OracleSGDS` must be available.

If the `OracleSGDS` data source is configured and available, SPARQL Gateway servlet will automatically create all the necessary tables and indexes upon initialization.

7.17.2.5 Add and Configure the SparqlGatewayAdminGroup Group, if Desired

The following JSP files in SPARQL Gateway can help you to view, edit, and update SPARQL queries and XSL transformations that are stored in an Oracle database:

```
http://<host>:7001/sparqlgateway/admin/sparql.jsp
http://<host>:7001/sparqlgateway/admin/xslt.jsp
```

These files are protected by HTTP Basic Authentication. In `WEB-INF/weblogic.xml`, a principal named `SparqlGatewayAdminGroup` is defined.

To be able to log in to either of these JSP pages, you must use the WebLogic Server to add a group named `SparqlGatewayAdminGroup`, and create a new user or assign an existing user to this group.

7.17.3 Using SPARQL Gateway with RDF Data

The primary interface for an application to interact with SPARQL Gateway is through a URL with the following format:

```
http://host:port/sparqlgateway/sg?<SPARQL_ENDPOINT>&<SPARQL_QUERY>&<XSLT>
```

In the preceding format:

- *<SPARQL_ENDPOINT>* specifies the `ee` parameter, which contains a URL encoded form of a SPARQL endpoint.

  For example, `ee=http%3A%2F%2Fsparql.org%2Fbooks` is the URL encoded string for SPARQL endpoint `http://sparql.org/books`. It means that SPARQL queries are to be executed against endpoint `http://sparql.org/books`.

- *<SPARQL_QUERY>* specifies either the SPARQL query, or the location of the SPARQL query.

  If it is feasible for an application to accept a very long URL, you can encode the whole SPARQL query and set `eq=<encoded_SPARQL_query>` in the URL If it is not feasible for an application to accept a very long URL, you can store the SPARQL queries and make them available to SPARQL Gateway using one of the approaches described in Storing SPARQL Queries and XSL Transformations.

- *<XSLT>* specifies either the XSL transformation, or the location of the XSL transformation.

  If it is feasible for an application to accept a very long URL, you can encode the whole XSL transformation and set `ex=<encoded_XSLT>` in the URL If it is not feasible for an application to accept a very long URL, you can store the XSL transformations and make them available to SPARQL Gateway using one of the approaches described in Storing SPARQL Queries and XSL Transformations.

7.17.3.1 Storing SPARQL Queries and XSL Transformations

If it is not feasible for an application to accept a very long URL, you can specify the location of the SPARQL query and the XSL transformation in the *<SPARQL_QUERY>* and *<XSLT>*

portions of the URL format described in Using SPARQL Gateway with Semantic Data, using any of the following approaches:

- Store the SPARQL queries and XSL transformations in the SPARQL Gateway Web application itself.

  To do this, unpack the `sparqlgateway.war` file, and store the SPARQL queries and XSL transformations in the top-level directory; then pack the `sparqlgateway.war` file and redeploy it.

  The `sparqlgateway.war` file includes the following example files: `qb1.sparql` (SPARQL query) and `default.xslt` (XSL transformation).

  > **Tip:**
  >
  > Use the file extension `.sparql` for SPARQL query files, and the file extension `.xslt` for XSL transformation files.

  The syntax for specifying these files (using the provided example file names) is **`wq`**`=qb1`.sparql for a SPARQL query file and **`wx`**`=default.xslt` for an XSL transformation file.

  If you want to customize the default XSL transformations, see the examples in Customizing the Default XSLT File.

  If you specify `wx=noop.xslt`, XSL transformation is not performed and the SPARQL response is returned "as is" to the client.

- Store the SPARQL queries and XSL transformations in a file system directory, and make sure that the directory is accessible for the deployed SPARQL Gateway Web application.

  By default, the directory is set to `/tmp`, as shown in the following `<init-param>` setting:

  ```
  <init-param>
     <param-name>sparql_gateway_repository_filedir</param-name>
     <param-value>/tmp/</param-value>
  </init-param>
  ```

  It is recommended that you customize this directory before deploying the SPARQL Gateway. To change the directory setting, edit the text in between the `<param-value>` and `</param-value>` tags.

  The following example specifies a SPARQL query file and an XSL transformation file that are in the directory specified in the `<init-param>` element for `sparql_gateway_repository_filedir`:

  ```
  fq=qb1.sparql
  fx=myxslt1.xslt
  ```

- Make the SPARQL queries and XSL transformations accessible from a website.

  By default, the website directory is set to `http://127.0.0.1/queries/`, as shown in the following `<init-param>` setting:

  ```
  <init-param>
     <param-name>sparql_gateway_repository_url</param-name>
     <param-value>http://127.0.0.1/queries/</param-value>
  </init-param>
  ```

Customize this directory before deploying the SPARQL Gateway. To change the website setting, edit the text in between the `<param-value>` and `</param-value>` tags.

The following example specifies a SPARQL query file and an XSL transformation file that are in the URL specified in the `<init-param>` element for `sparql_gateway_repository_url`.

```
uq=qb1.sparql
ux=myxslt1.xslt
```

Internally, SPARQL Gateway computes the appropriate complete URL, fetches the content, starts query execution, and applies the XSL transformation to the query response XML.

- Store the SPARQL queries and XSL transformations in an Oracle database.

  This approach requires that the J2EE data source `OracleSGDS` be defined. After SPARQL Gateway retrieves a database connection from the OracleSGDS data source, a SPARQL query is read from the database table ORACLE_ORARDF_SG_QUERY using the integer ID provided.

  The syntax for fetching a SPARQL query from an Oracle database is **dq=**`<integer-id>`, and the syntax for fetching an XSL transformation from an Oracle database is **dx=**`<integer-id>`.

  Upon servlet initialization, the following tables are created automatically if they do not already exist (you do not need to create them manually):

  – ORACLE_ORARDF_SG_QUERY with a primary key of QID (integer type)

  – ORACLE_ORARDF_SG_XSLT with a primary key of XID (integer type)

7.17.3.2 Specifying a Timeout Value

When you submit a potentially long-running query using the URL format described in Using SPARQL Gateway with Semantic Data, you can limit the execution time by specifying a timeout value in milliseconds. For example, the following shows the URL format and a timeout specification that the SPARQL query execution started from SPARQL Gateway is to be ended after 1000 milliseconds (1 second):

```
http://host:port/sparqlgateway/sg?<SPARQL_ENDPOINT>&<SPARQL_QUERY>&<XSLT>&t=1000
```

If a query does not finish when timeout occurs, then an empty SPARQL response is constructed by SPARQL Gateway.

Note that even if SPARQL Gateway times out a query execution at the HTTP connection level, the query may still be running on the server side. The actual behavior will be vendor-dependent.

7.17.3.3 Specifying Best Effort Query Execution

> **Note:**
>
> You can specify best effort query execution only if you also specify a timeout value (described in Specifying a Timeout Value).

When you submit a potentially long-running query using the URL format described in Using SPARQL Gateway with Semantic Data, if you specify a timeout value, you can also specify a "best effort" limitation on the query. For example, the following shows the URL format with a timeout specification of 1000 milliseconds (1 second) and a best effort specification (`&b=t`):

```
http://host:port/sparqlgateway/sg?<SPARQL_ENDPOINT>&<SPARQL_QUERY>&<XSLT>&t=1000&b=t
```

The web.xml file includes two parameter settings that affect the behavior of the best effort option: `sparql_gateway_besteffort_maxrounds` and `sparql_gateway_besteffort_maxthreads`. The following show the default definitions:

```
<init-param>
  <param-name>sparql_gateway_besteffort_maxrounds</param-name>
  <param-value>10</param-value>
</init-param>

<init-param>
  <param-name>sparql_gateway_besteffort_maxthreads</param-name>
  <param-value>3</param-value>
</init-param>
```

When a SPARQL SELECT query is executed in best effort style, a series of queries will be executed with an increasing LIMIT value setting in the SPARQL query body. (The core idea is based on the observation that a SPARQL query runs faster with a smaller LIMIT setting.) SPARQL Gateway starts query execution with a "LIMIT 1" setting. Ideally, this query can finish before the timeout is due. Assume that is the case, the next query will have its LIMIT setting is increased, and subsequent queries have higher limits. The maximum number of query executions is controlled by the `sparql_gateway_besteffort_maxrounds` parameter.

If it is possible to run the series of queries in parallel, the `sparql_gateway_besteffort_maxthreads` parameter controls the degree of parallelism.

7.17.3.4 Specifying a Content Type Other Than text/xml

By default, SPARQL Gateway assumes that XSL transformations generate XML, and so the default content type set for HTTP response is `text/xml`. However, if your application requires a response format other than XML, you can specify the format in an additional URL parameter (with syntax `&rt=`), using the following format:

```
http://host:port/sparqlgateway/sg?
<SPARQL_ENDPOINT>&<SPARQL_QUERY>&<XSLT>&rt=<content_type>
```

Note that *<content_type>* must be URL encoded.

7.17.4 Customizing the Default XSLT File

You can customize the default XSL transformation file (the one referenced using `wx=default.xslt`). This section presents some examples of customizations.

The following example implements this namespace prefix replacement logic: if a variable binding returns a URI that starts with `http://purl.org/goodrelations/v1#`, that portion is replaced by `gr:`; and if a variable binding returns a URI that starts with `http://www.w3.org/2000/01/rdf-schema#`, that portion is replaced by `rdfs:`.

```
<xsl:when test="starts-with(text(),'http://purl.org/goodrelations/v1#')">
   <xsl:value-of select="concat('gr:',substring-after(text(),'http://purl.org/
goodrelations/v1#'))"/>
</xsl:when>
...
```

```
<xsl:when test="starts-with(text(),'http://www.w3.org/2000/01/rdf-schema#')">
   <xsl:value-of select="concat('rdfs:',substring-after(text(),'http://www.w3.org/
2000/01/rdf-schema#'))"/>
</xsl:when>
```

The following example implements logic to trim a leading `http://localhost/` or a leading `http://127.0.0.1/`.

```
<xsl:when test="starts-with(text(),'http://localhost/')">
  <xsl:value-of select="substring-after(text(),'http://localhost/')"/>
</xsl:when>
<xsl:when test="starts-with(text(),'http://127.0.0.1/')">
  <xsl:value-of select="substring-after(text(),'http://127.0.0.1/')"/>
</xsl:when>
```

7.17.5 Using the SPARQL Gateway Java API

In addition to a Web interface, the SPARQL Gateway administration service provides a convenient Java application programming interface (API) for managing SPARQL queries and their associated XSL transformations. The Java API is included in the RDF Graph support for Apache Jena library, `sdordfclient.jar`.

Java API reference information is available in the `javadoc_sparqlgateway.zip` file that is included in the SPARQL Gateway .zip file (described in Download the RDF Graph Support for Apache Jena .zip File (if Not Already Done)).

The main entry point for this API is the `oracle.spatial.rdf.client.jena.SGDBHandler` class (SPARQL Gateway Database Handler), which provides the following static methods for managing queries and transformations:

- `deleteSparqlQuery(Connection, int)`
- `deleteXslt(Connection, int)`
- `insertSparqlQuery(Connection, int, String, String, boolean)`
- `insertXslt(Connection, int, String, String, boolean)`
- `getSparqlQuery(Connection, int, StringBuilder, StringBuilder)`
- `getXslt(Connection, int, StringBuilder, StringBuilder)`

These methods manipulate and retrieve entries in the SPARQL Gateway associated tables that are stored in an Oracle Database instance. To use these methods, the necessary associated tables must already exist. If the tables do not exist, deploy the SPARQL Gateway on a Web server and access a URL in the following format:

```
http://<host>:<port>/sparqlgateway/sg?
```

where *<host>* is the host name of the Web server and *<port>* is the listening port of the Web server. Accessing this URL will automatically create the necessary tables if they do not already exist.

Any changes made through the Java API affect the SPARQL Gateway Web service in the same way as changes made through the administration Web interface. This provides the flexibility to manage queries and transformations using the interface you find most convenient.

Note that the insert methods provided by the Java API will not replace existing queries or transformations stored in the tables. Attempting to replace an existing query or transformation will fail. To replace a query or transformation, you must remove the existing entry in the table using one of the delete methods, and then insert the new query or transformation using one of the insert methods.

The following examples demonstrate how to perform common management tasks using the Java API. The examples assume a connection has already been established to the underlying Oracle Database instance backing the SPARQL Gateway.

Example 7-33 Storing a SPARQL Query and an XSL Transformation

Example 7-33 adds a query and an XSL transformation to the database backing the SPARQL Gateway. After the query and transformation are added, other programs can use the query and transformation through the gateway by specifying the appropriate query ID (`qid`) and XSL transformation ID (`xid`) in the request URL.

Note that Although Example 7-33 inserts both a query and transformation, the query and transformation are not necessarily related and do not need to be used together when accessing SPARQL Gateway. Any query in the database can be used with any transformation in the database when submitting a request to SPARQL Gateway.

```
String query = "PREFIX ... SELECT ..."; // full SPARQL query text
String xslt  = "<?xml ...> ...";        // full XSLT transformation text

String queryDesc = "Conference attendee information"; // description of SPARQL query
String xsltDesc = "BIEE table widget transformation"; // description of XSLT
transformation

int queryId = queryIdCounter++; // assign a unique ID to this query
int xsltId  = xsltIdCounter++;  // assign a unique ID to this transformation

// Inserting a query or transformation will fail if the table already contains
// an entry with the same ID.  Setting this boolean to true will ignore these
// exceptions (but the table will remain unchanged). Here we specify that we
// want an exception thrown if we encounter a duplicate ID.
boolean ignoreDupException = false;

// add the query
try {
  // Delete query if one already exists with this ID (this will not throw an
  // error if no such entry exists)
  SGDBHandler.deleteSparqlQuery( connection, queryId );
  SGDBHandler.insertSparqlQuery( connection, queryId, query, queryDesc,
ignoreDupException );
} catch( SQLException sqle ) {
  // Handle exception
} catch( QueryException qe ) {
  // Handle query syntax exception
}

// add the XSLT
try {
  // Delete xslt if one already exists with this ID (this will not throw an
  // error if no such entry exists)
  SGDBHandler.deleteXslt( connection, xsltId );
  SGDBHandler.insertXslt( connection, xsltId, xslt, xsltDesc, ignoreDupException );
}  catch( SQLException sqle ) {
  // Handle database exception
} catch( TransformerConfigurationException tce ) {
  // Handle XSLT syntax exception
}
```

Example 7-34 Modifying a Query

Example 7-34 retrieves an existing query from the database, modifies it, then stores the updated version of the query back in the database. These steps simulate editing a query and saving the changes. (Note that if the query does not exist, an exception is thrown.)

```
StringBuilder query;
StringBuilder description;

// Populate these with the query text and description from the database
query = new StringBuilder( );
description = new StringBuilder( );

// Get the query from the database
try {
  SGDBHandler.getSparqlQuery( connection, queryId, query, description );
} catch( SQLException sqle ) {
  // Handle exception
  // NOTE: exception is thrown if query with specified ID does not exist
}

// The query and description should be populated now

// Modify the query
String updatedQuery = query.toString( ).replaceAll("invite", "attendee");

// Insert the query back into the database
boolean ignoreDup = false;
try {
  // First must delete the old query
  SGDBHandler.deleteSparqlQuery( connection, queryId );
  // Now we can add
  SGDBHandler.insertSparqlQuery( connection, queryId, updatedQuery,
description.toString( ), ignoreDup );
} catch( SQLException sqle ) {
  // Handle exception
} catch( QueryException qe ) {
  // Handle query syntax exception
}
```

Example 7-35 Retrieving and Printing an XSL Transformation

Example 7-35 retrieves an existing XSL transformation and prints it to standard output. (Note that if the transformation does not exist, an exception is thrown.)

```
StringBuilder xslt;
StringBuilder description;

// Populate these with the XSLT text and description from the database
xslt = new StringBuilder( );
description = new StringBuilder( );

try {
  SGDBHandler.getXslt( connection, xsltId, xslt, description );
} catch( SQLException sqle ) {
  // Handle exception
  // NOTE: exception is thrown if transformation with specified ID does not exist
}

// Print it to standard output
System.out.printf( "XSLT description: %s\n", description.toString( ) );
System.out.printf( "XSLT body:\n%s\n", xslt.toString( ) );
```

7.17.6 Using the SPARQL Gateway Graphical Web Interface

SPARQL Gateway provides several browser-based interfaces to help you test queries, navigate RDF data, and manage SPQARQL query and XSLT files.

- Main Page (index.html)
- Navigation and Browsing Page (browse.jsp)
- XSLT Management Page (xslt.jsp)
- SPARQL Management Page (sparql.jsp)

7.17.6.1 Main Page (index.html)

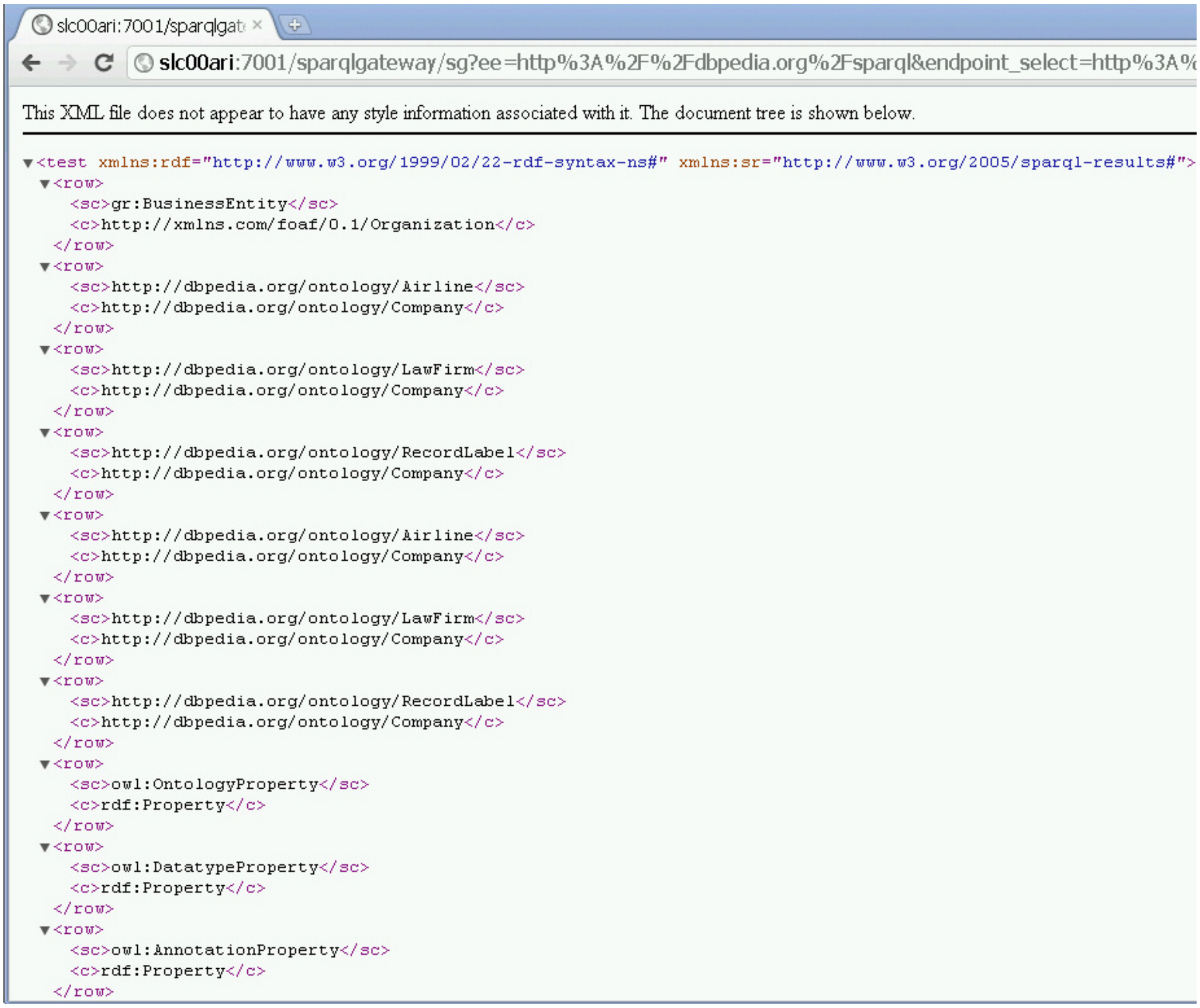
`http://<host>:<port>/sparqlgateway/index.html` provides a simple interface for executing SPARQL queries and then applying the transformations in the default.xslt file to the response. Figure 7-2 shows this interface for executing a query.

Figure 7-2   Graphical Interface Main Page (index.html)

Enter or select a **SPARQL Endpoint**, specify the **SPARQL SELECT Query Body,** and press **Submit Query**.

For example, if you specify `http://dbpedia.org/sparql` as the SPARQL endpoint and use the SPARQL query body from Figure 7-2, the response will be similar to Figure 7-3. Note that the default transformations (in `default.xslt`) have been applied to the XML output in this figure.

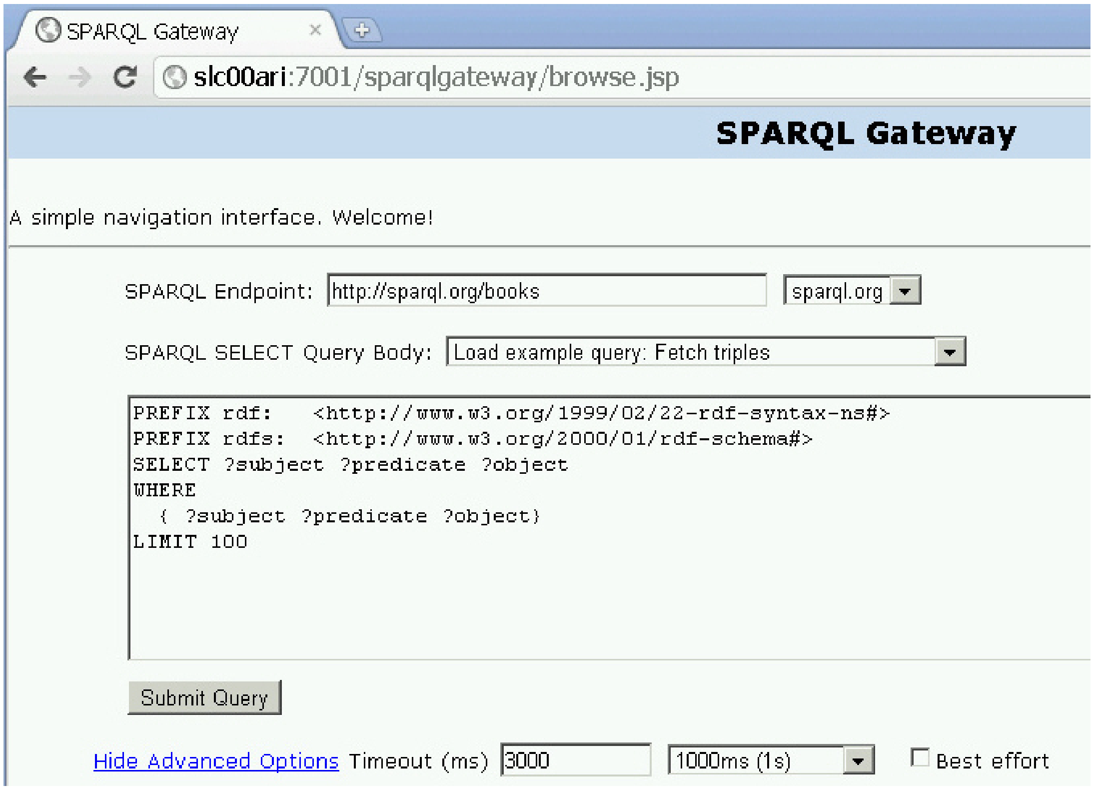
Figure 7-3 SPARQL Query Main Page Response



7.17.6.2 Navigation and Browsing Page (browse.jsp)

`http://<host>:<port>/sparqlgateway/browse.jsp` provides navigation and browsing capabilities for RDF data. It works against any standard compliant SPARQL endpoint. Figure 7-4 shows this interface for executing a query.

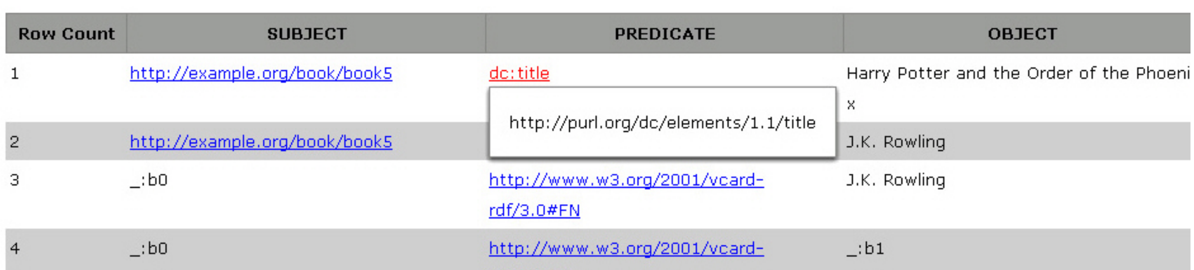
Figure 7-4 Graphical Interface Navigation and Browsing Page (browse.jsp)



Enter or select a **SPARQL Endpoint**, specify the **SPARQL SELECT Query Body,** optionally specify a **Timeout (ms)** value in milliseconds and the **Best Effort** option, and press **Submit Query**.

The SPARQL response is parsed and then presented in table form, as shown in Figure 7-5.

Figure 7-5 Browsing and Navigation Page: Response

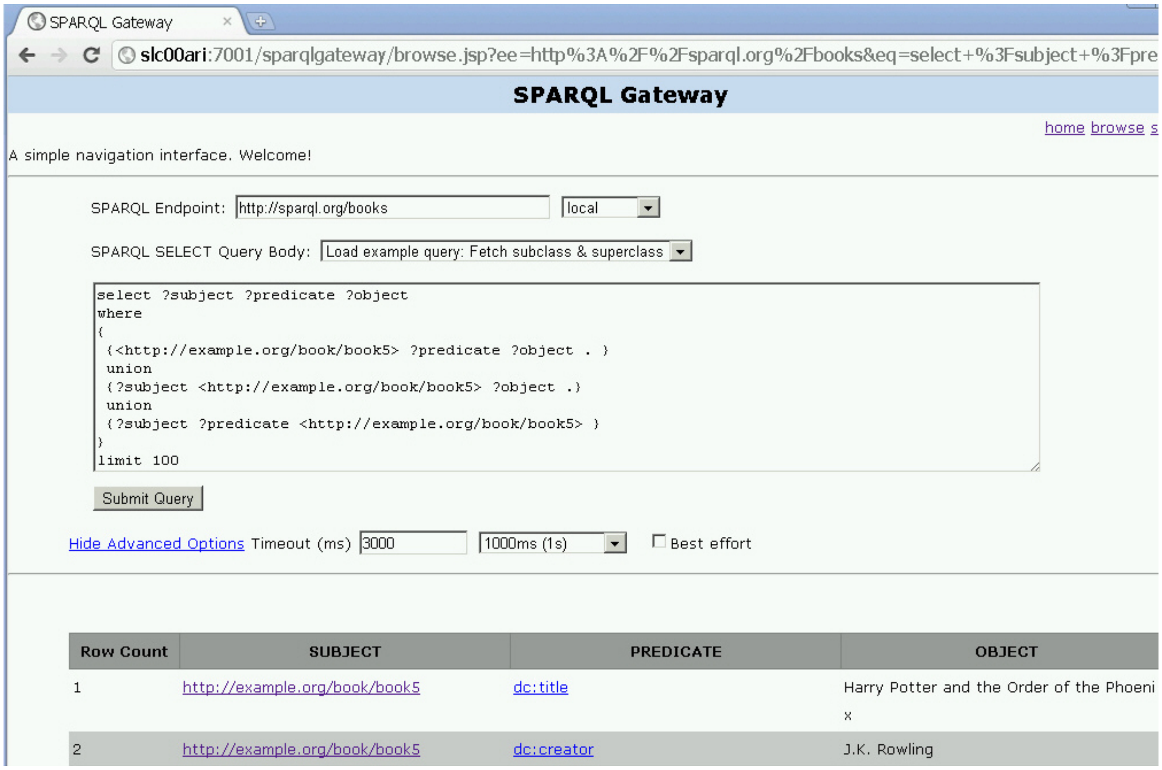


In Figure 7-5, note that URIs are clickable to allow navigation, and that when users move the cursor over a URI, tool tips are shown for the URIs which have been shortened for readability (as in `http://purl.org.dc/elements/1.1/title` being displayed as the tool tip for `dc:title` in the figure).

If you click the URI `http://example.org/book/book5` in the output shown in Figure 7-5, a new SPARQL query is automatically generated and executed. This generated SPARQL query has three query patterns that use this particular URI as subject, predicate, and object, as shown in

Figure 7-6. Such a query can give you a good idea about how this URI is used and how it is related to other resources in the data set.

Figure 7-6 Query and Response from Clicking URI Link



When there are many matches of a query, the results are organized in pages and you can click on any page. The page size by default is 50 results. To display more (or fewer) than 50 rows per page in a response with the Browsing and Navigation Page (`browse.jsp`), you can specify the `&resultsPerPage` parameter in the URL. For example, to allow 100 rows per page, include the following in the URL:

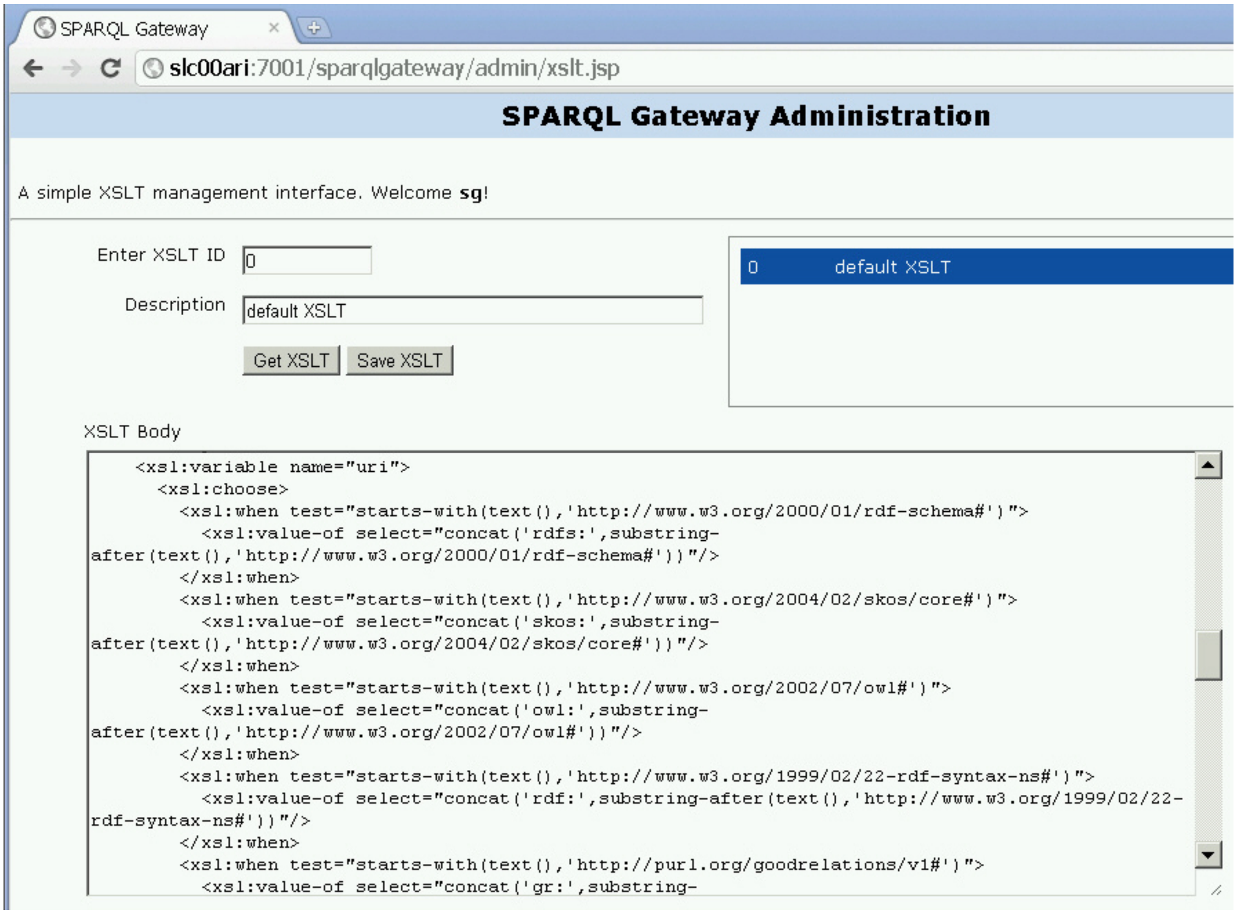
```
&resultsPerPage=100
```

7.17.6.3 XSLT Management Page (xslt.jsp)

`http://<host>:<port>/sparqlgateway/admin/xslt.jsp` provides a simple XSLT management interface. You can enter an XSLT ID (integer) and click **Get XSLT** to retrieve both the Description and XSLT Body. You can modify the XSLT Body text and then save the changes by clicking **Save XSLT**. Note that there is a previewer to help you navigate among available XSLT definitions.

Figure 7-7 shows the XSLT Management Page.

Figure 7-7 XSLT Management Page

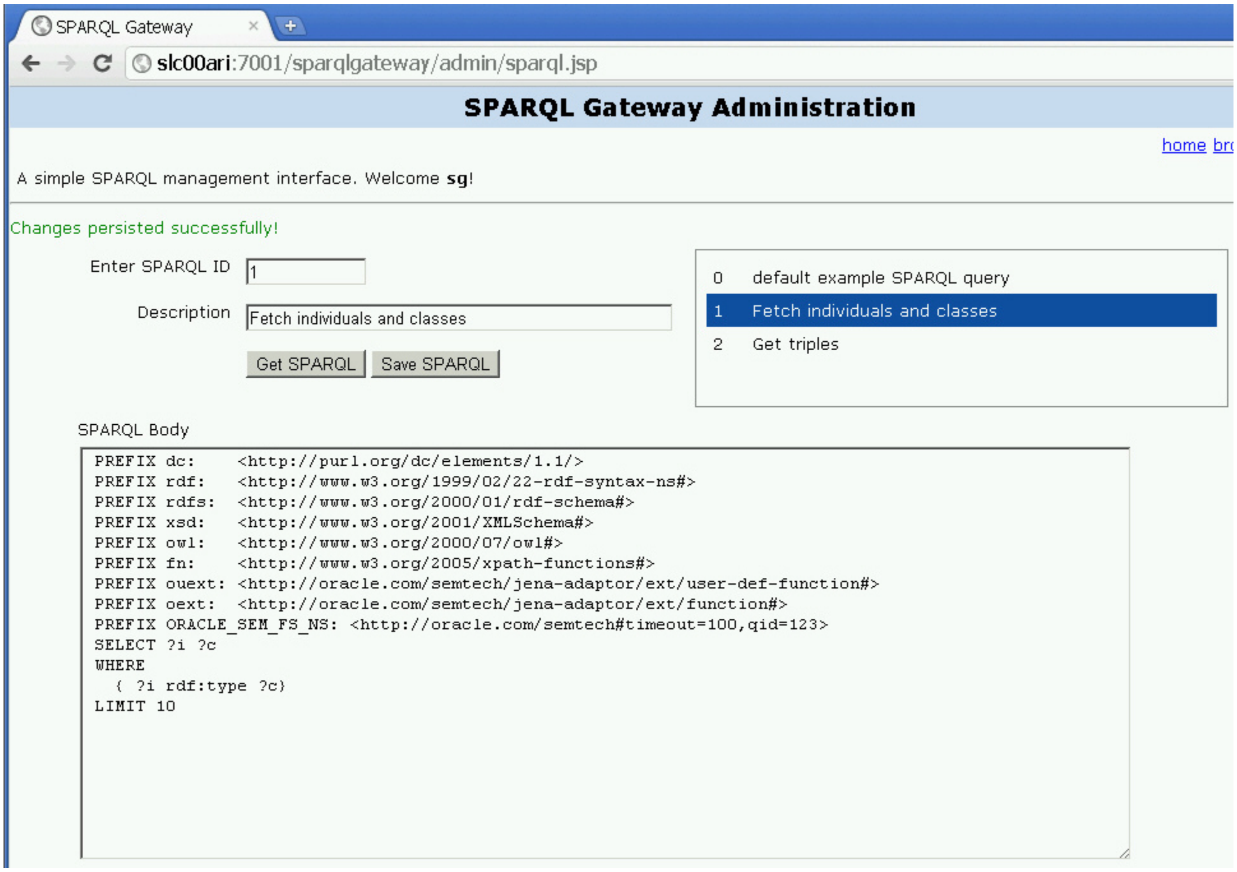


7.17.6.4 SPARQL Management Page (sparql.jsp)

`http://<host>:<port>/sparqlgateway/admin/xslt.jsp` provides a simple SPARQL management interface. You can enter a SPARQL ID (integer) and click **Get SPARQL** to retrieve both the Description and SPARQL Body. You can modify the SPARQL Body text and then save the changes by clicking **Save SPARQL**. Note that there is a previewer to help you navigate among available SPARQL queries.

Figure 7-8 shows the SPARQL Management Page.

Figure 7-8 SPARQL Management Page

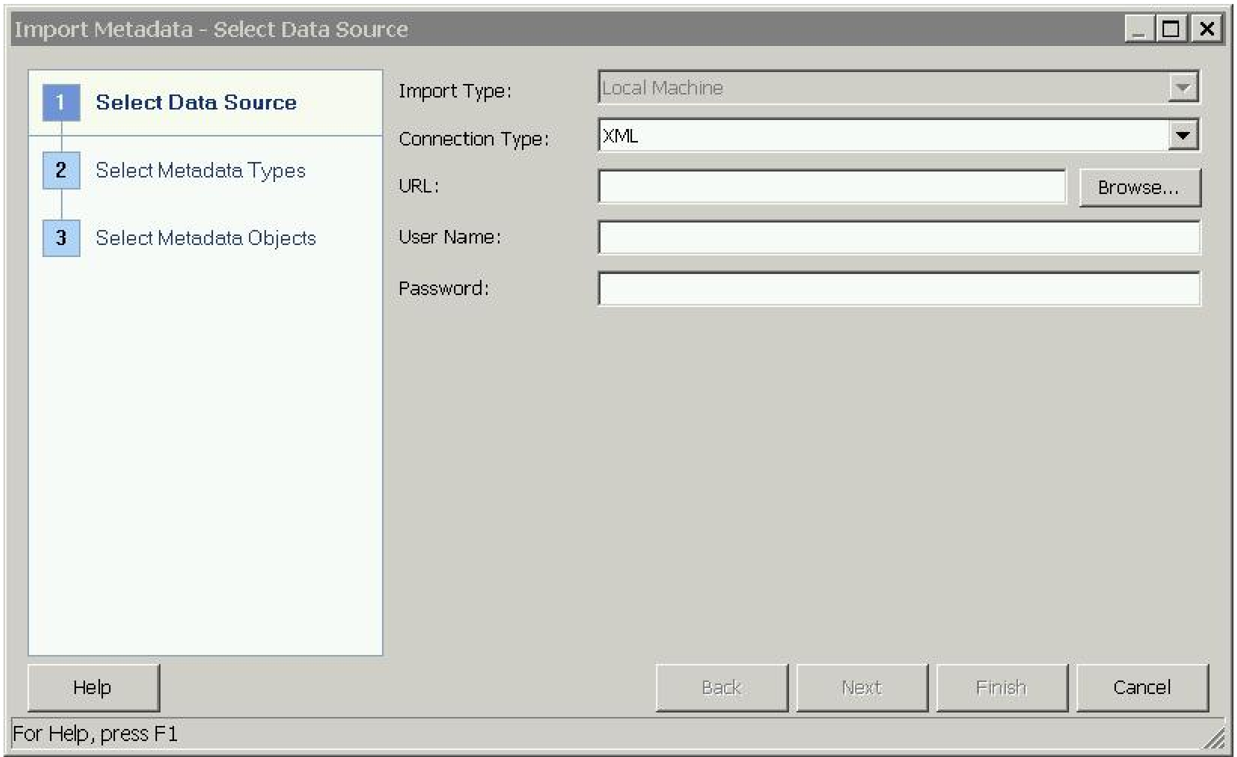


7.17.7 Using SPARQL Gateway as an XML Data Source to OBIEE

This section explains how to create an XML Data source for Oracle Business Intelligence Enterprise Edition (OBIEE), by integrating OBIEE with RDF using SPARQL Gateway as a bridge. (The specific steps and illustrations reflect the Oracle BI Administration Tool Version 11.1.1.3.0.100806.0408.000.)

1. Start the Oracle BI Administration Tool.
2. Click **File**, then **Import Metadata**. The first page of the Import Metadata wizard is displayed, as shown in Figure 7-9.

Figure 7-9 Import Metadata - Select Data Source



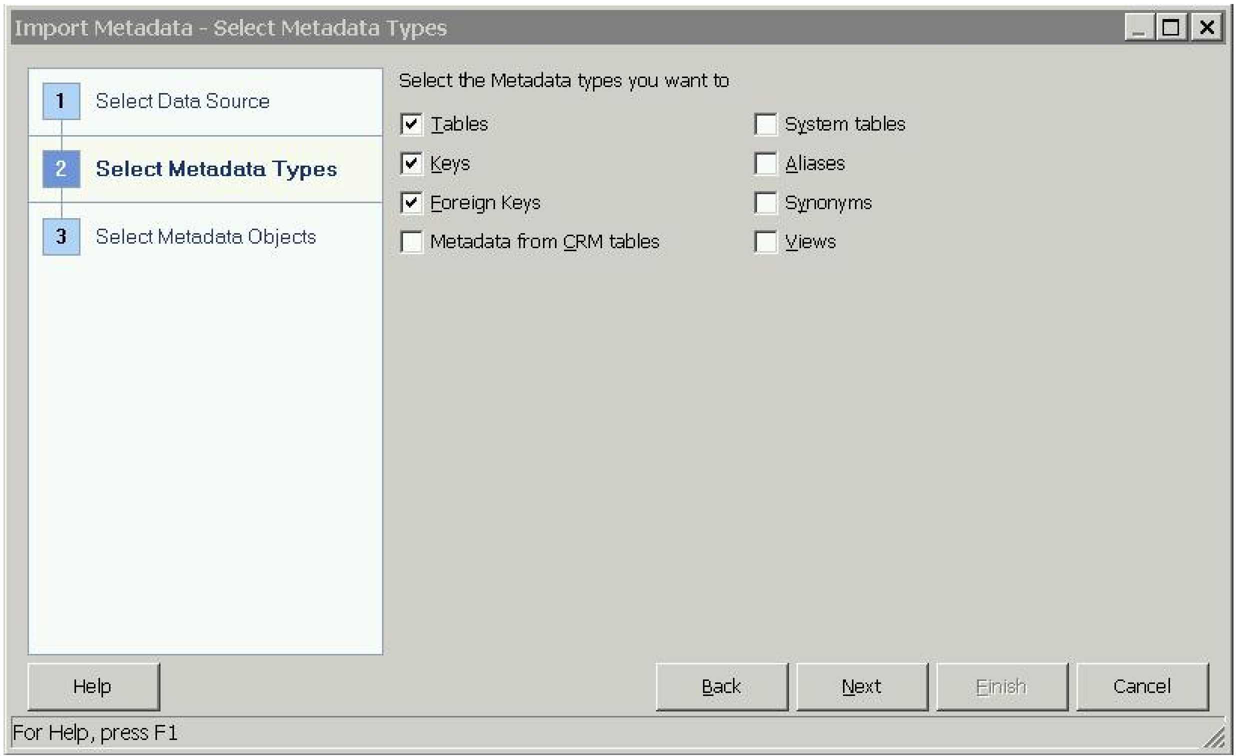
Connection Type: Select `XML`.

URL: URL for an application to interact with SPARQL Gateway, as explained in Using SPARQL Gateway with Semantic Data. You can also include the timeout and best effort options.

Ignore the **User Name** and **Password** fields.

3. Click **Next**. The second page of the Import Metadata wizard is displayed, as shown in Figure 7-10.

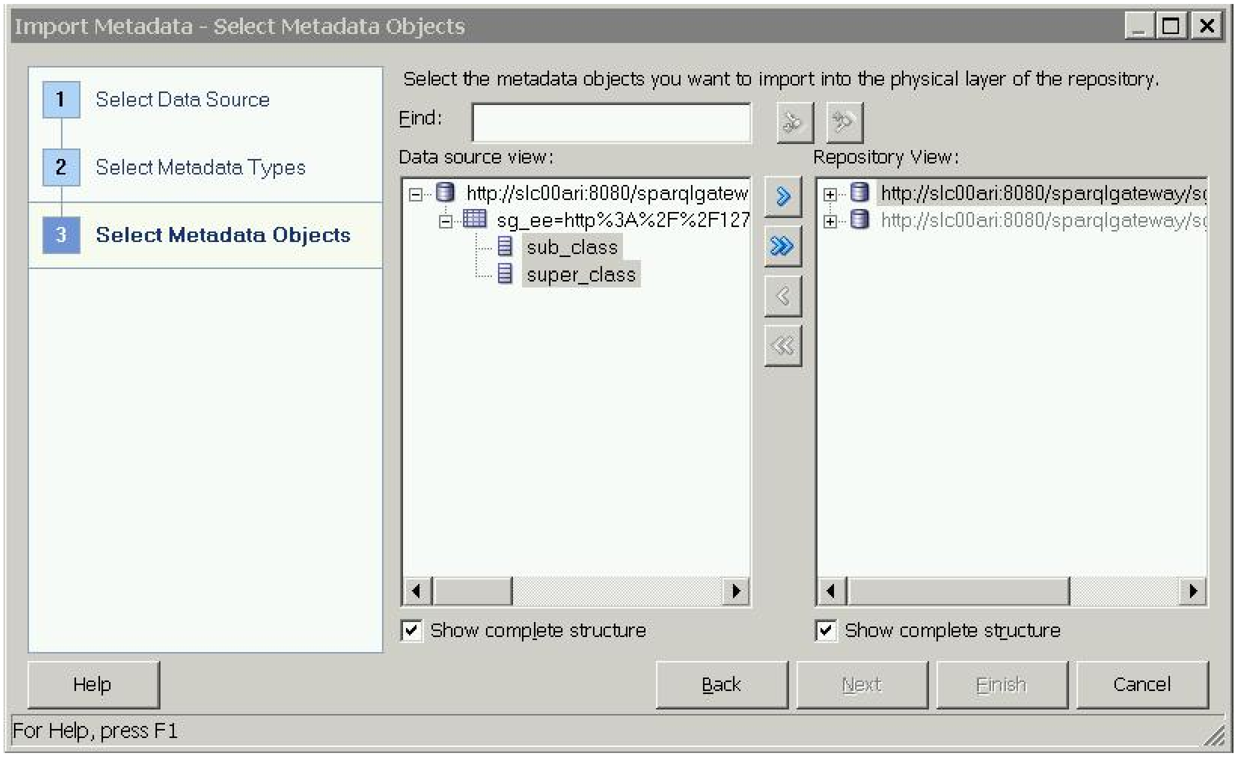
Figure 7-10 Import Metadata - Select Metadata Types



Select the desired metadata types to be imported. Be sure that **Tables** is included in the selected types.

4. Click **Next**. The third page of the Import Metadata wizard is displayed, as shown in Figure 7-11.

Figure 7-11 Import Metadata - Select Metadata Objects



In the **Data Source View**, expand the node that has the table icon, select the column names (mapped from projected variables defined in the SPARQL SELECT statement), and click the right-arrow (**>**) button to move the selected columns to the **Repository View**.

5. Click **Finish**.
6. Complete the remaining steps for the usual BI Business Model work and Mapping and Presentation definition work, which are not specific to SPARQL Gateway or RDF data.

7.18 Deploying Fuseki in Apache Tomcat

To deploy Fuseki in Apache Tomcat, you can use the Tomcat admin web page, or you can just copy the Fuseki `.war` file into the `webapps` folder of Tomcat and it will be automatically deployed.

This topic describe the auto-deploy steps. It assumes that the $FUSEKI_BASE setup is done and the configuration files exist (by default, Fuseki uses `/etc/fuseki` as the directory to store its configuration files).

1. Download and install the latest version of Apache Tomcat.

   The directory root for Apache Tomcat installation will be referred to in these instructions as $CATALINA_HOME.
2. Copy the `fuseki.war` into the Tomcat webapps folder. For example:

```
cd $CATALINA_HOME/webapps
cp /tmp/jena_adapter/fuseki_web_app/fuseki.war  .
```

3. Start Tomcat:

```
$CATALINA_HOME/bin/startup.sh
```

If this file does not have executable permission, enter the following command and then again attempt to start Tomcat:

```
chmod u+x $CATALINA_HOME/bin/startup.sh
```

4. In a browser go to: `http://hostname:8080/fuseki`

7.19 ORARDFLDR Utility for Bulk Loading RDF Data

This section describes using the ORARDFLDR utility program for Bulk Loading RDF Data.

This utility program loads all files in a directory into an RDF graph in Oracle database. It supports several RDF serializations like RDF/XML, Turtle, N-Triple, N-Quads and Trig. Files compressed with gzip can be directly loaded without uncompressing the gzip file. In addition, Unicode character escaping and long literals (CLOBs) are handled automatically.

Running ORARDFLDR Utility Program

The following describes the commands to execute ORARDFLDR:

Prerequisite: Ensure that the environment variable `${ORACLE_JENA_HOME}` is pointing to the directory where the OTN kit is stored.

Usage:

```
java -cp ${ORACLE_JENA_HOME}/jar/'*' oracle.spatial.rdf.client.jena.utilities.RDFLoader
<command_line_arguments>
```

For help details:

```
java -cp ${ORACLE_JENA_HOME}/jar/'*' oracle.spatial.rdf.client.jena.utilities.RDFLoader
--help
```

For convenience, a shell script in the bin directory can also be executed. The following describes the commands to use this script

Prerequisite: Set `${ORACLE_JENA_HOME}` and ensure `${ORACLE_JENA_HOME}/bin` is in your Unix `PATH` environment variable.

Usage:

```
orardfldr <command_line_arguments>
```

For help details:

```
orardfldr --help
```

- Using ORARDFLDR with Oracle Autonomous Database

7.19.1 Using ORARDFLDR with Oracle Autonomous Database

This section describes using the ORARDFLDR utility with Oracle Autonomous Database.

The ORARDFLDR utility included with support for Apache Jena can be used to bulk load RDF files from your client computer to Oracle Autonomous Database. The connection with the database is based on a cloud wallet.

General instructions for connecting to an Oracle Autonomous Database with JDBC can be found in Java connectivity to ATP.

The following example describes establishing a JDBC connection to Oracle Autonomous Database following the Plain JDBC using JKS files procedure.

Example 7-36 JDBC connectivity to Oracle Autonomous Database

Prerequisite: Ensure you have the following Oracle jar files: `ojdbc8.jar`, `ucp.jar`, `oraclepki.jar`, `osdt_core.jar`, and `osdt_cert.jar`.

1. Unzip your `wallet_<dbname>.zip` file. You should see something similar to the listing below after unzipping the file.

```
[oracle@localhost Wallet_Info]$ ls
cwallet.sso  keystore.jks      README      tnsnames.ora
ewallet.p12  ojdbc.properties  sqlnet.ora  truststore.jks
```

2. Modify `ojdbc.properties` to add JKS related connection properties. The final version of your ojdbc.properties file should be similar as shown below:

```
# Connection property while using Oracle wallets.
#oracle.net.wallet_location=(SOURCE=(METHOD=FILE)(METHOD_DATA=(DIRECTORY=$
{TNS_ADMIN})))
# FOLLOW THESE STEPS FOR USING JKS
# (1) Uncomment the following properties to use JKS.
# (2) Comment out the oracle.net.wallet_location property above
# (3) Set the correct password for both trustStorePassword and keyStorePassword.
# It's the password you specified when downloading the wallet from OCI Console or
the Service Console.
javax.net.ssl.trustStore=${TNS_ADMIN}/truststore.jks
javax.net.ssl.trustStorePassword=password
javax.net.ssl.keyStore=${TNS_ADMIN}/keystore.jks
javax.net.ssl.keyStorePassword=password
```

Use the following JDBC URL:

```
jdbc:oracle:thin:@dbname_alias?TNS_ADMIN=<path_to_wallet_directory>
```

The following examples loads the RDF files using the ORAFLDR utility for a database named `rdfdb` and having a wallet directory as `/home/oracle/RDF/Wallet_Info/` .

Example 7-37 Using ORAFLDR Utility to load RDF Data files

Prerequisite: Ensure you have copied the prerequisite jars listed in Example 7-36 to `$ORACLE_JENA_HOME/jar/`.

Invoke `ORARDFLDR` to load RDF files from your client computer to an Autonomous database.

```
orardfldr --modelName=M1 --fileDir=./data --lang=N-TRIPLE
    --jdbcUrl=jdbc:oracle:thin:@rdfdb_medium?TNS_ADMIN=/home/oracle/RDF/Wallet_Info/
    --user="RDFUSER" --password=password --networkOwner="RDFUSER" --networkName=NET1
```

It loads RDF data in N-Triple format into a model named M1 in a network named NET1 owned by RDFUSER. RDFUSER is also used for the database connection.

8 RDF Graph Support for Eclipse RDF4J

Oracle RDF Graph Adapter for Eclipse RDF4J utilizes the popular Eclipse RDF4J framework to provide Java developers support to use the RDF graph feature of Oracle Database.

> **Note:**
>
> - This feature was previously referred to as the *Sesame Adapter for Oracle Database* and the *Sesame Adapter*.
> - Some optional features of RDF graph support for Eclipse RDF4J are supported only if Oracle JVM is enabled on your Oracle Autonomous Database Serverless deployments. To enable Oracle JVM, see Use Oracle Java in *Using Oracle Autonomous Database Serverless* for more information. Specifically, database stored procedure based execution of SPARQL-to-SQL query translation and SPARQL Update require Oracle JVM. These options are further discussed in SPARQL Query Execution Model and SPARQL Update Execution Model.

The Eclipse RDF4J is a powerful Java framework for processing and handling RDF data. This includes creating, parsing, scalable storage, reasoning and querying with RDF and Linked Data. See https://rdf4j.org for more information.

This chapter assumes that you are familiar with major concepts explained in RDF Graph Overview and OWL Concepts . It also assumes that you are familiar with the overall capabilities and use of the Eclipse RDF4J Java framework. See https://rdf4j.org for more information.

The Oracle RDF Graph Adapter for Eclipse RDF4J extends the RDF data management capabilities of Oracle Database RDF/OWL by providing a popular standards based API for Java developers.

- Oracle RDF Graph Support for Eclipse RDF4J Overview
  The Oracle RDF Graph Adapter for Eclipse RDF4J API provides a Java-based interface to Oracle RDF data through an API framework and tools that adhere to the Eclipse RDF4J SAIL API.
- Prerequisites for Using Oracle RDF Graph Adapter for Eclipse RDF4J
  Before you start using the Oracle RDF Graph Adapter for Eclipse RDF4J, you must ensure that your system environment meets certain prerequisites.
- Setup and Configuration for Using Oracle RDF Graph Adapter for Eclipse RDF4J
  To use the Oracle RDF Graph Adapter for Eclipse RDF4J, you must first setup and configure the system environment.
- Using Oracle RDF Graph Adapter for Eclipse RDF4J with Oracle Autonomous Database
  To use Oracle RDF Graph Adapter for Eclipse RDF4J with Autonomous Database, you can use a JDBC connection to connect to your Autonomous Database instance.
- Database Connection Management
  The Oracle RDF Graph Adapter for Eclipse RDF4J provides support for Oracle Database Connection Pooling.

8.1 Oracle RDF Graph Support for Eclipse RDF4J Overview

The Oracle RDF Graph Adapter for Eclipse RDF4J API provides a Java-based interface to Oracle RDF data through an API framework and tools that adhere to the Eclipse RDF4J SAIL API.

The RDF Graph support for Eclipse RDF4J is similar to the RDF Graph support for Apache Jena as described in RDF Graph Support for Apache Jena.

The adapter for Eclipse RDF4J provides a Java API for interacting with RDF data stored in Oracle Database. It also provides integration with the following Eclipse RDF4J tools:

- Eclipse RDF4J Server, which provides an HTTP SPARQL endpoint.
- Eclipse RDF4J Workbench, which is a web-based client UI for managing databases and executing queries.

The features provided by the adapter for Eclispe RDF4J include:

- Loading (bulk and incremental), exporting, and removing statements, with and without context
- Querying data, with and without context
- Updating data, with and without context
- Validating data with SHACL constraints

Oracle RDF Graph Adapter for Eclipse RDF4J implements various interfaces of the Eclipse RDF4J Storage and Inference Layer (SAIL) API.

For example, the class `OracleSailConnection` is an Oracle implementation of the Eclipse RDF4J `SailConnection` interface, and the class `OracleSailStore` extends `AbstractSail` which is an Oracle implementation of the Eclipse RDF4J `Sail` interface.

The following example demonstrates a typical usage flow for the RDF Graph support for Eclipse RDF4J.

Example 8-1 Sample Usage flow for RDF Graph Support for Eclipse RDF4J Using a Schema-Private RDF Network

```
String networkOwner = "SCOTT";
String networkName = "NET1";
String modelName = "UsageFlow";
OraclePool oraclePool = new OraclePool(jdbcurl, user, password);
SailRepository sr = new SailRepository(new OracleSailStore(oraclePool, modelName,
networkOwner, networkName));
SailRepositoryConnection conn = sr.getConnection();

//A ValueFactory factory for creating IRIs, blank nodes, literals and statements
ValueFactory vf = conn.getValueFactory();
IRI alice = vf.createIRI("http://example.org/Alice");
IRI friendOf = vf.createIRI("http://example.org/friendOf");
IRI bob = vf.createIRI("http://example.org/Bob");
Resource context1 = vf.createIRI("http://example.org/");

// Data loading can happen here.
conn.add(alice, friendOf, bob, context1);
String query =
  " PREFIX foaf: <http://xmlns.com/foaf/0.1/> " +
  " PREFIX dc: <http://purl.org/dc/elements/1.1/> " +
  " select ?s ?p ?o ?name WHERE {?s ?p ?o . OPTIONAL {?o foaf:name ?name .} } ";
TupleQuery tq = conn.prepareTupleQuery(QueryLanguage.SPARQL, query);
TupleQueryResult tqr = tq.evaluate();
while (tqr.hasNext()) {
    System.out.println((tqr.next().toString()));
}
tqr.close();
conn.close();
sr.shutDown();
```

8.2 Prerequisites for Using Oracle RDF Graph Adapter for Eclipse RDF4J

Before you start using the Oracle RDF Graph Adapter for Eclipse RDF4J, you must ensure that your system environment meets certain prerequisites.

The following are the prerequisites required for using the adapter for Eclipse RDF4J:

- Oracle Database Standard Edition 2 (SE2) or Enterprise Edition (EE) for version 18c or later (user managed database in the cloud or on-premise), or Oracle Autonomous Database version 19c or later.
- Eclipse RDF4J version 4.3.14
- JDK 11

8.3 Setup and Configuration for Using Oracle RDF Graph Adapter for Eclipse RDF4J

To use the Oracle RDF Graph Adapter for Eclipse RDF4J, you must first setup and configure the system environment.

The adapter can be used in the following three environments:

- Programmatically through Java code
- Accessed over HTTP as a SPARQL Service
- Used within the Eclipse RDF4J workbench environment

The following sections describe the actions for using the adapter for Eclipse RDF4J in the above mentioned environments:

- Setting up Oracle RDF Graph Adapter for Eclipse RDF4J for Use with Java
- Setting Up Oracle RDF Graph Adapter for Eclipse RDF4J for Use in RDF4J Server and Workbench
- Setting Up Oracle RDF Graph Adapter for Eclipse RDF4J for Use As SPARQL Service

8.3.1 Setting up Oracle RDF Graph Adapter for Eclipse RDF4J for Use with Java

To use the Oracle RDF Graph Adapter for Eclipse RDF4J programmatically through Java code, you must first ensure that the system environment meets all the prerequisites as explained in Prerequisites for Using Oracle RDF Graph Adapter for Eclipse RDF4J.

Before you can start using the adapter to store, manage, and query RDF graphs in the Oracle database, you need to create an RDF network. An RDF network acts like a folder that can hold multiple RDF graphs created by database users. RDF networks can be created in a user schema (referred to as a schema-private network).

A network can be created by invoking the following command:

```
sem_apis.create_RDF_network(<tablespace_name>, network_owner=><network_owner>,
network_name=><network_name>)
```

See RDF Networks for more information.

> **See Also:**
>
> - Setting up Oracle RDF Graph Adapter for Eclipse RDF4J for Use with Java for Oracle Database 19c and later
> - Setting up Oracle RDF Graph Adapter for Eclipse RDF4J for Use with Java for Oracle Database 18c

Creating a Schema-Private RDF Network

You can create a schema-private RDF network by performing the following actions from a SQL based interface such as SQL Developer, SQLPLUS, or from a Java program using JDBC:

1. Connect to **Oracle Database** as a `SYSTEM` user with a DBA privilege (or as `ADMIN` user on Autonomous Database Serverless).

   ```
   CONNECT system/<password-for-system-user>
   ```

2. Create a **tablespace** for storing the **user data**. Use a suitable operating system folder and filename.
   If you are using Autonomous Database Serverless, then use the pre-created `DATA` tablespace.

   ```
   CREATE TABLESPACE usertbs
     DATAFILE 'usertbs.dat'
     SIZE 128M REUSE
     AUTOEXTEND ON NEXT 64M
     MAXSIZE UNLIMITED
     SEGMENT SPACE MANAGEMENT AUTO;
   ```

3. Create a database **user** to create and own the RDF network. This user can create or use RDF graphs or do both within this schema-private network using the adapter.

   ```
   CREATE USER rdfuser
     IDENTIFIED BY <password-for-rdfuser>
     DEFAULT TABLESPACE usertbs
     QUOTA 5G ON usertbs;
   ```

4. Grant the necessary **privileges** to the new database user.

   ```
   GRANT CONNECT, RESOURCE, CREATE VIEW TO rdfuser;
   ```

5. Connect to **Oracle Database** as `rdfuser`.

   ```
   CONNECT rdfuser/<password-for-rdf-user>
   ```

6. Create a schema-private **RDF network** named `NET1`.

   ```
   EXECUTE SEM_APIS.CREATE_RDF_NETWORK(tablespace_name =>'usertbs',
   network_owner=>'RDFUSER', network_name=>'NET1');
   ```

7. Verify that schema-private RDF network has been created successfully.

   ```
   SELECT table_name
     FROM sys.all_tables
     WHERE table_name = 'NET1#RDF_VALUE$' AND owner='RDFUSER';
   ```

   Presence of `<NETWORK_NAME>#RDF_VALUE$` table in the network owner's schema shows that the schema-private RDF network has been created successfully.

   ```
   TABLE_NAME
   -----------
   NET1#RDF_VALUE$
   ```

You can now set up the Oracle RDF Graph Adapter for Eclipse RDF4J for use with Java code by performing the following actions:

1. Download and configure Eclipse RDF4J Release 4.3.14 from RDF4J Downloads page.
2. Download the adapter for Eclipse RDF4J, (Oracle Adapter for Eclipse RDF4J) from Oracle Software Delivery Cloud.
3. Unzip the downloaded kit (V1047295-01.zip) into a temporary directory, such as `/tmp/oracle_adapter`, on a Linux system. If this temporary directory does not already exist, create it before the unzip operation.

4. Include the Oracle Adapter for Eclipse RDF4J jar files from the `/jar` directory of the kit in your `CLASSPATH`:
   - `oracle-rdf4j-adapter-4.3.14-20250106.jar`
   - `sdordf-23.6.0-20241122.jar`
   - `sdordf-client-23.6.0-20241122.jar`
   - `sdoutl-23.6.0-20241122.jar`
5. Include the following supporting libraries in your `CLASSPATH`, in order to run your Java code through your IDE:
   - `eclipse-rdf4j-4.3.14-onejar.jar`: Download this Eclipse RDF4J jar library from RDF4J Downloads page.
   - `ojdbc8.jar`: Download this JDBC thin driver for your database version from JDBC Downloads page.
   - `ucp.jar`: Download this Universal Connection Pool jar file for your database version from JDBC Downloads page.
   - `log4j-api-2.24.2.jar`, `log4j-core-2.24.2.jar`, `log4j-slf4j-impl-2.24.2.jar`, `slf4j-api-1.7.36.jar`, and `commons-io-2.14.0.jar`: Download from Apache Software Foundation.
6. If you want to use `JSON-LD` support, include the following additional libraries in you `CLASSPATH`:
   - `httpclient-4.5.14.jar`, `httpclient-cache-4.5.14.jar`, `httpclient-osgi-4.5.13.jar`: Download from Apache Software Foundation.
   - `httpcore-4.4.16.jar`, `httpcore-nio-4.4.14.jar`, `httpcore-osgi-4.4.14.jar`: Download from Apache Software Foundation.
   - `jackson-annotations-2.13.5.jar`, `jackson-core-2.13.5.jar`, `jackson-databind-2.13.5.jar`: Download from GitHub
   - `jsonld-java-0.13.4.jar`: Download from GitHub
   - `rdf4j-rio-jsonld-4.3.14.jar`, `rdf4j-rio-rdfjson-4.3.14.jar`: Download from RDF4J Downloads
7. Install JDK 11 if it is not already installed.
8. Set the `JAVA_HOME` environment variable to refer to the JDK 11 installation. Define and verify the setting by executing the following command:

```
echo $JAVA_HOME
```

8.3.2 Setting Up Oracle RDF Graph Adapter for Eclipse RDF4J for Use in RDF4J Server and Workbench

This section describes the installation and configuration of the Oracle RDF Graph Adapter for Eclipse RDF4J in RDF4J Server and RDF4J Workbench.

The RDF4J Server is a database management application that provides HTTP access to RDF4J repositories, exposing them as SPARQL endpoints. RDF4J Workbench provides a web interface for creating, querying, updating and exploring the repositories of an RDF4J Server.

Prerequisites

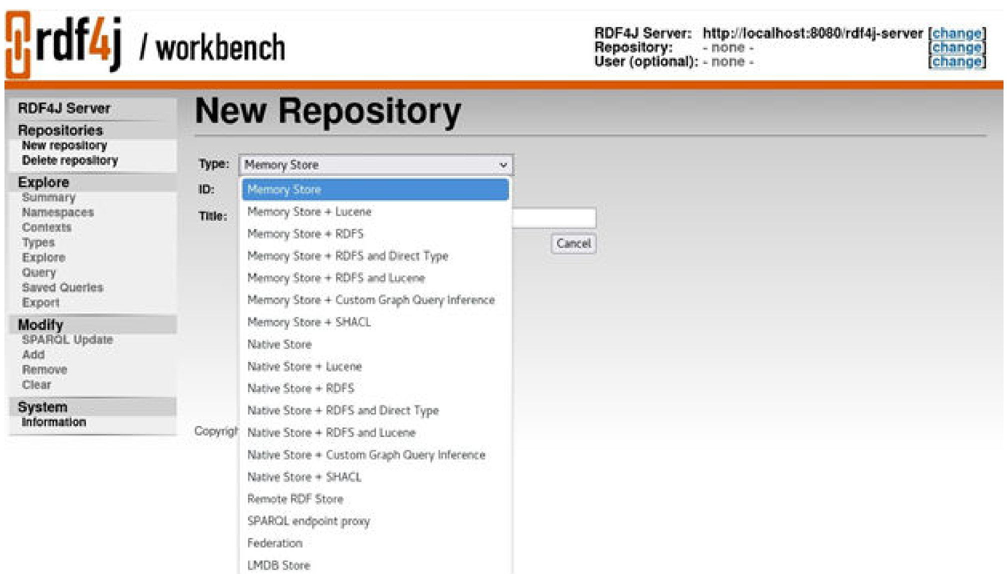
Ensure the following prerequisites are configured to use the adapter for Eclipse RDF4J in RDF4J Server and Workbench:

1. Java 11 runtime environment.
2. Download the supporting libraries as explained in Include Supporting Libraries.
3. A Java Servlet Container that supports Java Servlet API 4.0 and Java Server Pages (JSP) 2.3, or newer.

> **Note:**
>
> All examples in this chapter are executed on a recent, stable version of Apache Tomcat (9.0.97).

4. Standard installation of the RDF4J Server, RDF4J Workbench, and RDF4J Console . See RDF4J Server and Workbench Installation and RDF4J Console installation for more information.
5. Verify that *Oracle* is not listed as a default repository in the drop-down in the following Figure 8-1.

Figure 8-1 Data Source Repository in RDF4J Workbench



> **Note:**
>
> If the Oracle data source repository is already set up in the RDF4J Workbench repository, then it will appear in the preceding drop-down list.

Adding the Oracle Data Source Repository in RDF4J Workbench

To add the Oracle data source repository in RDF4J Workbench, you must execute the following steps:

1. Add the **Data Source** to `context.xml` in Tomcat main `$CATALINA_HOME/conf/context.xml` directory, by updating the following highlighted fields.

```
- Using JDBC driver
    <Resource name="jdbc/OracleSemDS" auth="Container"
       driverClassName="oracle.jdbc.OracleDriver"
       factory="oracle.jdbc.pool.OracleDataSourceFactory"
       scope="Shareable"
       type="oracle.jdbc.pool.OracleDataSource"
       user="<<username>>"
       password="<<pwd>>"
       url="jdbc:oracle:thin:@<< host:port:sid >>"
       maxActive="100"
       minIdle="15"
       maxIdel="15"
       initialSize="15"
       removeAbandonedTimeout="30"
       validationQuery="select 1 from dual"
    />

- Using UCP
   <Resource name="jdbc/OracleSemDS" auth="Container"
       factory="oracle.ucp.jdbc.PoolDataSourceImpl"
       type="oracle.ucp.jdbc.PoolDataSource"
       connectionFactoryClassName="oracle.jdbc.pool.OracleDataSource"
       minPoolSize="15"
       maxPoolSize="100"
       inactiveConnectionTimeout="60"
       abandonedConnectionTimeout="30"
       initialPoolSize="15"
       user="<<username>>"
       password="<<pwd>>"
       url="jdbc:oracle:thin:@<< host:port:sid >>"
    />
```

2. Copy **Oracle jdbc** and **ucp driver** to Tomcat `lib` folder.

```
cp -f ojdbc8.jar $CATALINA_HOME/lib
cp -f ucp.jar $CATALINA_HOME/lib
```

3. Copy the `oracle-rdf4j-adapter-4.3.14-20250106.jar`, `sdordf-23.6.0-20241122.jar`, `sdordf-client-23.6.0-20241122.jar`, `sdoutl-23.6.0-20241122.jar` to RDF4J Server `lib` folder.

```
cp -f oracle-rdf4j-adapter-4.3.14-20250106.jar $CATALINA_HOME/webapps/rdf4j-server/
WEB-INF/lib
cp -f sdordf-23.6.0-20241122.jar $CATALINA_HOME/webapps/rdf4j-server/WEB-INF/lib
cp -f sdordf-client-23.6.0-20241122.jar $CATALINA_HOME/webapps/rdf4j-server/WEB-
INF/lib
cp -f sdoutl-23.6.0-20241122.jar $CATALINA_HOME/webapps/rdf4j-server/WEB-INF/lib
```

4. Copy the `oracle-rdf4j-adapter-4.3.14-20250106.jar`, `sdordf-23.6.0-20241122.jar`, `sdordf-client-23.6.0-20241122.jar`, `sdoutl-23.6.0-20241122.jar` to RDF4J Workbench `lib` folder.

```
cp -f oracle-rdf4j-adapter-4.3.14-20250106.jar $CATALINA_HOME/webapps/rdf4j-
workbench/WEB-INF/lib
cp -f sdordf-23.6.0-20241122.jar $CATALINA_HOME/webapps/rdf4j-workbench/WEB-INF/lib
cp -f sdordf-client-23.6.0-20241122.jar $CATALINA_HOME/webapps/rdf4j-workbench/WEB-
INF/lib
cp -f sdoutl-23.6.0-20241122.jar $CATALINA_HOME/webapps/rdf4j-workbench/WEB-INF/lib
```

5. Create the configuration file `create-oracle.xsl` within the Tomcat `$CATALINA_HOME/webapps/rdf4j-workbench/transformations` folder.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:sparql="http://www.w3.org/2005/sparql-results#"
  xmlns="http://www.w3.org/1999/xhtml">
  <xsl:include href="../locale/messages.xsl" />
  <xsl:variable name="title">
  <xsl:value-of select="$repository-create.title" />
  </xsl:variable>
  <xsl:include href="template.xsl" />
  <xsl:template match="sparql:sparql">

    <form action="create" method="post">
      <table class="dataentry">
        <tbody>
          <tr>
            <th>
              <xsl:value-of select="$repository-type.label" />
            </th>
            <td>
              <select id="type" name="type">
                <option value="oracle"> Oracle Sail Store </option>
              </select>
            </td>
            <td></td>
          </tr>
          <tr>
            <th>
              <xsl:value-of select="$repository-id.label" />
            </th>
            <td>
              <input type="text" id="id" name="Repository ID"
                size="16" value="native" />
            </td>
            <td></td>
          </tr>
          <tr>
            <th>
              <xsl:value-of select="$repository-title.label" />
            </th>
            <td>
              <input type="text" id="title" name="Repository title" size="50"
                value="Native store" />
            </td>
            <td></td>
          </tr>
          <tr>
            <th> Model Name <strong style="color:Red">*</strong> </th>
            <td>
              <input type="text" id="oracleSemModelName" name="Model Name" size="50"
value="" />
            </td>
            <td></td>
          </tr>
          <tr>
            <th> Network Owner </th>
            <td>
              <input type="text" id="oracleSemNetworkOwner" name="Network Owner"
```

```
size="50" value="" />
            </td>
            <td></td>
          </tr>
          <tr>
            <th> Network Name </th>
            <td>
              <input type="text" id="oracleSemNetworkName" name="Network Name"
size="50" value="" />
            </td>
            <td></td>
          </tr>
          <tr>
            <th> DataSource Name </th>
            <td>
              <input type="text" id="oracleSemDataSourceName" name="DataSource Name"
size="30" value="OracleSemDS" />
            </td>
            <td></td>
          </tr>
          <tr>
            <td></td>
            <td>
              <input type="button" value="{$cancel.label}"
                  style="float:right" data-href="repositories"
                  onclick="document.location.href=this.getAttribute('data-href')" />
              <input id="create" type="button" value="{$create.label}"
                  onclick="checkOverwrite()" />
            </td>
          </tr>
        </tbody>
      </table>
    </form>
  <script src="../../scripts/create.js" type="text/javascript"></script>
  </xsl:template>
</xsl:stylesheet>
```

6. Replace the configuration file `create.xsl` within the Tomcat `$CATALINA_HOME/webapps/rdf4j-workbench/transformations` transformation folder.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE xsl:stylesheet [
   <!ENTITY xsd  "http://www.w3.org/2001/XMLSchema#" >
 ]>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:sparql="http://www.w3.org/2005/sparql-results#"
  xmlns="http://www.w3.org/1999/xhtml">

  <xsl:include href="../locale/messages.xsl" />

  <xsl:variable name="title">
    <xsl:value-of select="$repository-create.title" />
  </xsl:variable>

  <xsl:include href="template.xsl" />

  <xsl:template match="sparql:sparql">
    <form action="create">
      <table class="dataentry">
        <tbody>
          <tr>
            <th>
```

```
      <xsl:value-of select="$repository-type.label" />
    </th>
    <td>
      <select id="type" name="type">
        <option value="memory">
          Memory Store
        </option>
        <option value="memory-lucene">
          Memory Store + Lucene
        </option>
        <option value="memory-rdfs">
          Memory Store + RDFS
        </option>
        <option value="memory-rdfs-dt">
          Memory Store + RDFS and Direct Type
        </option>
        <option value="memory-rdfs-lucene">
          Memory Store + RDFS and Lucene
        </option>
        <option value="memory-customrule">
          Memory Store + Custom Graph Query Inference
        </option>
        <option value="memory-shacl">
          Memory Store + SHACL
        </option>
        <option value="native">
          Native Store
        </option>
        <option value="native-lucene">
          Native Store + Lucene
        </option>
        <option value="native-rdfs">
          Native Store + RDFS
        </option>
        <option value="native-rdfs-dt">
          Native Store + RDFS and Direct Type
        </option>
        <option value="memory-rdfs-lucene">
          Native Store + RDFS and Lucene
        </option>
        <option value="native-customrule">
          Native Store + Custom Graph Query Inference
        </option>
        <option value="native-shacl">
          Native Store + SHACL
        </option>
        <option value="remote">
          Remote RDF Store
        </option>
        <option value="sparql">
          SPARQL endpoint proxy
        </option>
        <option value="federate">Federation</option>
        <option value="lmdb">LMDB Store</option>
        <option value="oracle">Oracle</option>
      </select>
    </td>
    <td></td>
  </tr>
  <tr>
    <th>
      <xsl:value-of select="$repository-id.label" />
```

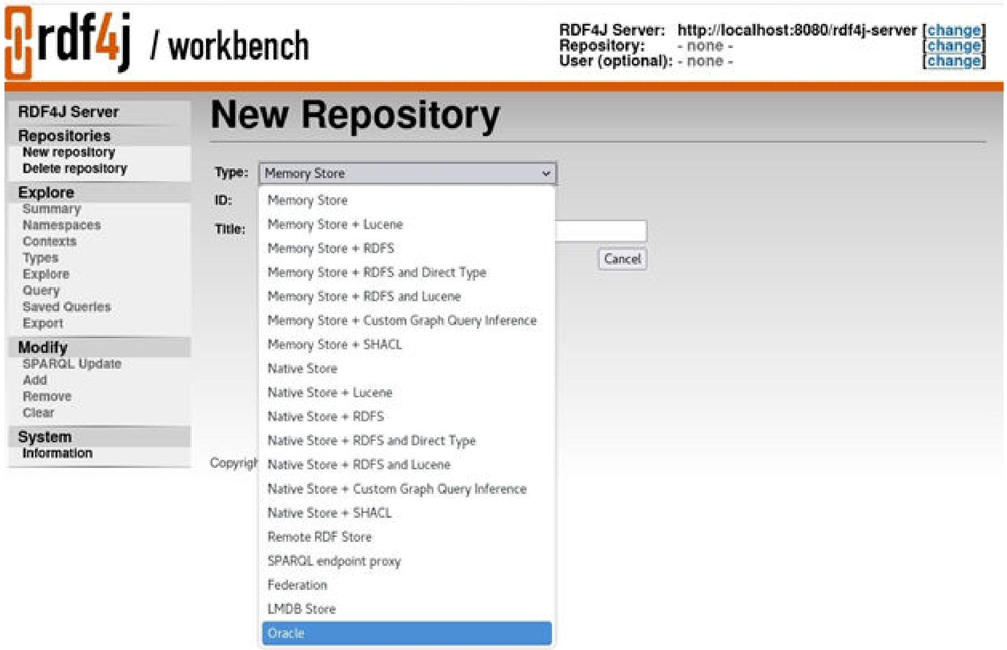
```
              </th>
              <td>
                <input type="text" id="id" name="id" size="16" />
              </td>
              <td></td>
            </tr>
            <tr>
              <th>
                <xsl:value-of select="$repository-title.label" />
              </th>
              <td>
                <input type="text" id="title" name="title" size="48" />
              </td>
              <td></td>
            </tr>
            <tr>
              <td></td>
              <td>
                <input type="button" value="{$cancel.label}" style="float:right"
                  data-href="repositories"
                  onclick="document.location.href=this.getAttribute('data-href')" />
                <input type="submit" name="next" value="{$next.label}" />
              </td>
            </tr>
          </tbody>
        </table>
      </form>
    </xsl:template>

  </xsl:stylesheet>
```

7. Restart Tomcat and navigate to `https://localhost:8080/rdf4j-workbench`.

> **Note:**
>
> The configuration files, `create-oracle.xsl` and `create.xsl` contain the word "*Oracle*", which you can see in the drop-down in Figure 8-2

"*Oracle*" appears as an option in the drop-down list in RDF4J Workbench.

Figure 8-2 RDF4J Workbench Repository



- Using the Adapter for Eclipse RFD4J Through RDF4J Workbench
  You can use RDF4J Workbench for creating and querying repositories.

8.3.2.1 Using the Adapter for Eclipse RFD4J Through RDF4J Workbench

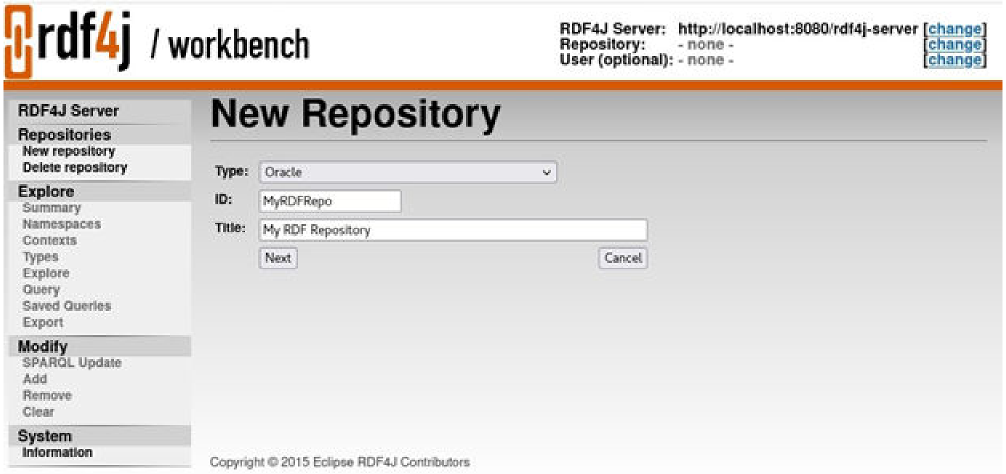
You can use RDF4J Workbench for creating and querying repositories.

RDF4J Workbench provides a web interface for creating, querying, updating and exploring repositories in RDF4J Server.

Creating a New Repository using RDF4J Workbench

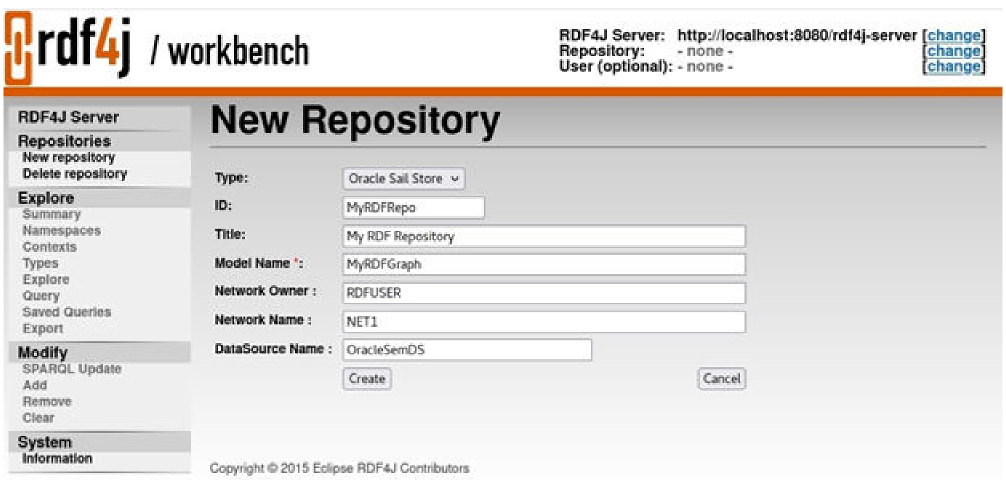
1. Start **RDF4J Workbench** by entering the url `https://localhost:8080/rdf4j-workbench` in your browser.
2. Click **New Repository** in the sidebar menu and select the new repository **Type** as "*Oracle*".
3. Enter the new repository **ID** and **Title** as shown in the following figure and click **Next**.

Figure 8-3 RDF4J Workbench New Repository



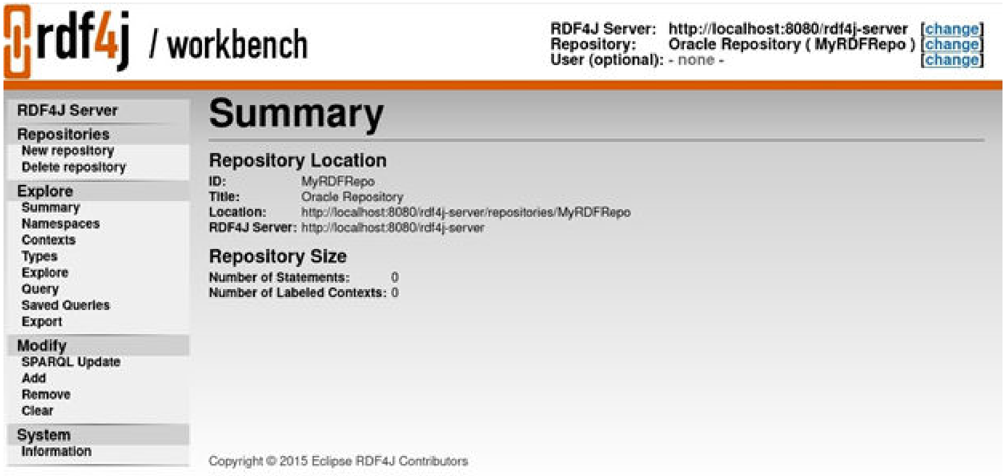
4. Enter your **Model (RDF graph)** details and click **Create** to create the new repository.

Figure 8-4 Create New Repository in RDF4J Workbench



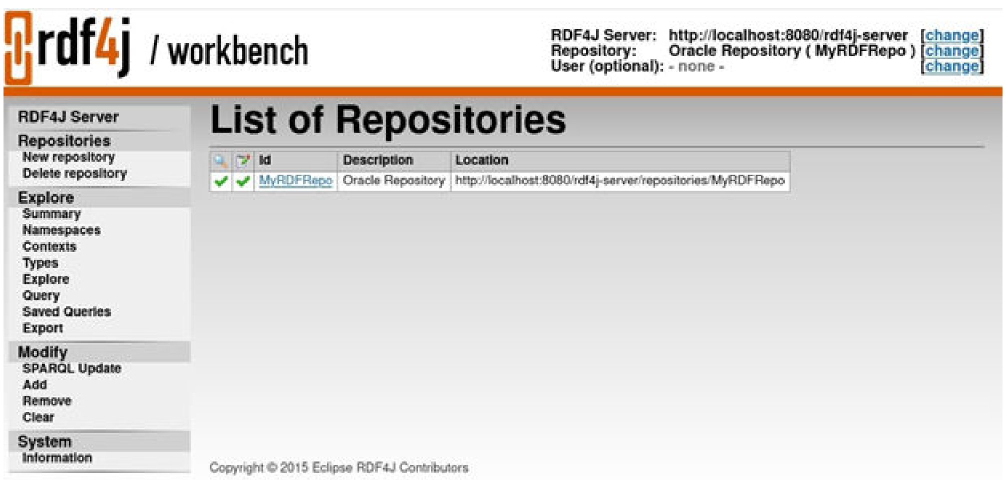
The newly created repository summary is display as shown:

Figure 8-5 Summary of New Repository in RDF4J Workbench



You can also view the newly created repository in the **List of Repositories** page in RDF4J Workbench.

Figure 8-6 List of Repositories



8.3.3 Setting Up Oracle RDF Graph Adapter for Eclipse RDF4J for Use As SPARQL Service

In order to use the SPARQL service via the RDF4J Workbench, ensure that the Eclipse RDF4J server is installed and the Oracle Data Source repository is configured as explained in Setting Up Oracle RDF Graph Adapter for Eclipse RDF4J for Use in RDF4J Server and Workbench

The Eclipse RDF4J server installation provides a REST API that uses the HTTP Protocol and covers a fully compliant implementation of the SPARQL 1.1 Protocol W3C Recommendation. This ensures that RDF4J server functions as a fully standards-compliant SPARQL endpoint. See The RDF4J REST API for more information on this feature.

The following section presents the examples of usage:

- Using the Adapter Over SPARQL Endpoint in Eclipse RDF4J Workbench

8.3.3.1 Using the Adapter Over SPARQL Endpoint in Eclipse RDF4J Workbench

This section provides a few examples of using the adapter for Eclipse RDF4J through a SPARQL Endpoint served by the Eclipse RDF4J Workbench.

Example 8-2 Request to Perform a SPARQL Update

The following example inserts some simple triples using HTTP POST. Assume that the content of the file `sparql_update.rq` is as follows:

```
PREFIX ex: <http://example.oracle.com/>
INSERT DATA {
  ex:a ex:value "A" .
  ex:b ex:value "B" .
}
```

You can then run the preceding SPARQL update using the `curl` command line tool as shown:

```
curl -X POST --data-binary "@sparql_update.rq" \
-H "Content-Type: application/sparql-update" \
"http://localhost:8080/rdf4j-server/repositories/MyRDFRepo/statements"
```

Example 8-3 Request to Execute a SPARQL Query Using HTTP GET

This `curl` example executes a SPARQL query using HTTP GET.

```
curl -X GET -H "Accept: application/sparql-results+json" \
"http://localhost:8080/rdf4j-server/repositories/MyRDFRepo?
query=SELECT%20%3Fs%20%3Fp%20%3Fo%0AWHERE%20%7B%20%3Fs%20%3Fp%20%3Fo%20%7D%0ALIMIT%2010"
```

Assuming that the previous SPARQL update example was executed on an empty repository, this REST request should return the following response.

```
{
  "head" : {
    "vars" : [
      "s",
      "p",
      "o"
    ]
  },
  "results" : {
    "bindings" : [
      {
        "p" : {
          "type" : "uri",
          "value" : "http://example.oracle.com/value"
        },
        "s" : {
          "type" : "uri",
          "value" : "http://example.oracle.com/b"
        },
        "o" : {
          "type" : "literal",
          "value" : "B"
        }
      },
      {
        "p" : {
          "type" : "uri",
```

```
            "value" : "http://example.oracle.com/value"
          },
          "s" : {
            "type" : "uri",
            "value" : "http://example.oracle.com/a"
          },
          "o" : {
            "type" : "literal",
            "value" : "A"
          }
        }
      ]
    }
  }
```


8.4 Using Oracle RDF Graph Adapter for Eclipse RDF4J with Oracle Autonomous Database

To use Oracle RDF Graph Adapter for Eclipse RDF4J with Autonomous Database, you can use a JDBC connection to connect to your Autonomous Database instance.

Refer to Connect with JDBC Thin Driver in *Using Oracle Autonomous Database Serverless* for more information on how to obtain the JDBC URL to connect to the Autonomous Database.

8.5 Database Connection Management

The Oracle RDF Graph Adapter for Eclipse RDF4J provides support for Oracle Database Connection Pooling.

Instances of `OracleSailStore` use a connection pool to manage connections to an Oracle database. Oracle Database Connection Pooling is provided through the `OraclePool` class. Usually, `OraclePool` is initialized with a `DataSource`, using the `OraclePool (DataSource ods)` constructor. In this case, `OraclePool` acts as an extended wrapper for the `DataSource`, while using the connection pooling capabilities of the data source. When you create an `OracleSailStore` object, it is sufficient to specify the `OraclePool` object in the store constructor, the database connections will then be managed automatically by the adapter for Eclipse RDF4J. Several other constructors are also provided for `OraclePool`, which, for example, allow you to create an `OraclePool` instance using a `JDBC URL` and database `username` and `password`. See the Javadoc included in the Oracle RDF Graph Adapter for Eclipse RDF4J download for more details.

If you need to retrieve Oracle connection objects (which are essentially database connection wrappers) explicitly, you can invoke the `OraclePool.getOracle` method. After finishing with the connection, you can invoke the `OraclePool.returnOracleDBtoPool` method to return the object to the connection pool.

When you get an `OracleSailConnection` from `OracleSailStore` or an `OracleSailRepositoryConnection` from an `OracleRepository`, a new `OracleDB` object is obtained from the `OraclePool` and used to create the RDF4J connection object. `READ_COMMITTED` transaction isolation is maintained between different RDF4J connection objects.

The one exception to this behavior occurs when you obtain an `OracleSailRepositoryConnection` by calling the `asRepositoryConnection` method on an existing instance of `OracleSailConnection`. In this case, the original `OracleSailConnection` and the newly obtained `OracleSailRepositoryConnection` will use the same OracleDB object. When you finish using an `OracleSailConnection` or `OracleSailRepositoryConnection` object, you should call its `close` method to return the `OracleDB` object to the `OraclePool`. Failing to do so will result in connection leaks in your application.

8.6 SPARQL Query Execution Model

SPARQL queries executed through the Oracle RDF Graph Adapter for Eclipse RDF4J API run as SQL queries against Oracle's relational schema for storing RDF data.

Utilizing Oracle's SQL engine allows SPARQL query execution to take advantage of many performance features such as parallel query execution, in-memory columnar representation, and Exadata smart scan.

There are two ways to execute a SPARQL query:

- You can obtain an implementation of `Query` or one of its subinterfaces from the `prepareQuery` functions of a `RepositoryConnection` that has an underlying `OracleSailConnection`.
- You can obtain an Oracle-specific implementation of `TupleExpr` from `OracleSPARQLParser` and call the `evaluate` method of `OracleSailConnection`.

The following code snippet illustrates the first approach.

```
//run a query against the repository
String queryString =
  "PREFIX ex: <http://example.org/ontology/>\n" +
  "SELECT * WHERE {?x ex:name ?y} LIMIT 1 ";
TupleQuery tupleQuery = conn.prepareTupleQuery(QueryLanguage.SPARQL, queryString);

try (TupleQueryResult result = tupleQuery.evaluate()) {
  while (result.hasNext()) {
    BindingSet bindingSet = result.next();
    psOut.println("value of x: " + bindingSet.getValue("x"));
    psOut.println("value of y: " + bindingSet.getValue("y"));
  }
}
```

When an `OracleSailConnection` evaluates a query, it first translates the SPARQL query to an equivalent SQL query, which is then executed on the database server. By default, the query is translated using Java code on the client. However, this SPARQL-to-SQL translation can instead use the `SEM_APIS.SPARQL_TO_SQL` stored procedure on the database server if the system property `oracle.rdf4j.adapter.sparqlInClient` is set to the value `F`. Using `SEM_APIS.SPARQL_TO_SQL` reduces database roundtrips and may result in better performance when there is a high latency between the client and database server. The results of the SQL query are processed and returned through one of the standard RDF4J query result interfaces.

- Using BIND Values
- Using JDBC BIND Values
- Additions to the SPARQL Query Syntax to Support Other Features
- Special Considerations for SPARQL Query Support

8.6.1 Using BIND Values

Oracle RDF Graph Adapter for Eclipse RDF4J supports bind values through the standard RDF4J bind value APIs, such as the `setBinding` procedures defined on the Query interface.

Oracle implements bind values by adding `SPARQL BIND` clauses to the original SPARQL query string.

For example, consider the following SPARQL query:

```
SELECT * WHERE { ?s <urn:fname> ?fname }
```

In the above query, you can set the value `<urn:john>` for the query variable `?s`. The tansformed query in that case would be:

```
SELECT * WHERE { BIND (<urn:john> AS ?s) ?s <urn:fname> ?fname }
```

> **Note:**
>
> This approach is subject to the standard variable scoping rules of SPARQL. So query variables that are not visible in the outermost graph pattern, such as variables that are not projected out of a subquery, cannot be replaced with bind values.

8.6.2 Using JDBC BIND Values

Oracle RDF Graph Adapter for Eclipse RDF4J allows the use of JDBC bind values in the underlying SQL statement that is executed for a SPARQL query. The JDBC bind value implementation is much more performant than the standard RDF4J bind value support described in the previous section.

JDBC bind value support uses the standard RDF4J `setBinding` API, but bind variables must be declared in a specific way, and a special query option must be passed in with the `ORACLE_SEM_SM_NS` namespace prefix. To enable JDBC bind variables for a query, you must include `USE_BIND_VAR=JDBC` in the `ORACLE_SEM_SM_NS` namespace prefix (for example, `PREFIX ORACLE_SEM_SM_NS: <http://oracle.com/semtech#USE_BIND_VAR=JDBC>`). When a SPARQL query includes this query option, all query variables that appear in a simple SPARQL BIND clause will be treated as JDBC bind values in the corresponding SQL query. A simple SPARQL BIND clause is one with the form `BIND (<constant> as ?var)`, for example `BIND("dummy" AS ?bindVar1)`.

The following code snippet illustrates how to use JDBC bind values.

Example 8-4 Using JDBC Bind Values

```
// query that uses USE_BIND_VAR=JDBC option and declares ?name as a JDBC bind
variable
String queryStr =
  "PREFIX ex: <http://example.org/>\n"+
  "PREFIX foaf: <http://xmlns.com/foaf/0.1/>\n"+
  "PREFIX ORACLE_SEM_SM_NS: <http://oracle.com/semtech#USE_BIND_VAR=JDBC>\n"+
  "SELECT ?friend\n" +
  "WHERE {\n" +
  "  BIND(\"\" AS ?name)\n" +
  "  ?x foaf:name ?name\n" +
  "  ?x foaf:knows ?y\n" +
  "  ?y foaf:name ?friend\n" +
  "}";

// prepare the TupleQuery with JDBC bind var option
TupleQuery tupleQuery = conn.prepareTupleQuery(QueryLanguage.SPARQL,
```

```
queryStr);

// find friends for Jack
tupleQuery.setBinding("name", vf.createLiteral("Jack");

try (TupleQueryResult result = tupleQuery.evaluate()) {
  while (result.hasNext()) {
    BindingSet bindingSet = result.next();
    System.out.println(bindingSet.getValue("friend").stringValue());
  }
}

// find friends for Jill
tupleQuery.setBinding("name", vf.createLiteral("Jill");

try (TupleQueryResult result = tupleQuery.evaluate()) {
  while (result.hasNext()) {
    BindingSet bindingSet = result.next();
    System.out.println(bindingSet.getValue("friend").stringValue());
  }
}
```

Note:

The JDBC bind value capability of Oracle RDF Graph Adapter for Eclipse RDF4J utilizes the bind variables feature of SEM_APIS.SPARQL_TO_SQL described in Using Bind Variables with SEM_APIS.SPARQL_TO_SQL.

- Limitations for JDBC Bind Value Support

8.6.2.1 Limitations for JDBC Bind Value Support

Only SPARQL SELECT and ASK queries support JDBC bind values.

The following are the limitations for JDBC bind value support:

- JDBC bind values are not supported in:
  - SPARQL CONSTRUCT queries
  - DESCRIBE queries
  - SPARQL Update statements
- Long RDF literal values of more than 4000 characters in length cannot be used as JDBC bind values.
- Blank nodes cannot be used as JDBC bind values.

8.6.3 Additions to the SPARQL Query Syntax to Support Other Features

The Oracle RDF Graph Adapter for Eclipse RDF4J allows you to pass in options for query generation and execution. It implements these capabilities by overloading the SPARQL namespace prefix syntax by using Oracle-specific namespaces that contain query options. The namespaces are in the form `PREFIX ORACLE_SEM_xx_NS`, where `xx` indicates the type of feature (such as `SM - SEM_MATCH`).

- Query Execution Options
- SPARQL_TO_SQL (SEM_MATCH) Options

8.6.3.1 Query Execution Options

You can pass query execution options to the database server by including a SPARQL `PREFIX` of the following form:

```
PREFIX ORACLE_SEM_FS_NS: <http://oracle.com/semtech#option>
```

The `option` in the above SPARQL `PREFIX` reflects a query option (or multiple options separated by commas) to be used during query execution.

The following options are supported:

- DOP=n: specifies the degree of parallelism (n) to use during query execution.
- ODS=n: specifies the level of optimizer dynamic sampling to use when generating an execution plan.

The following example query uses the `ORACLE_SEM_FS_NS` prefix to specify that a degree of parallelism of 4 should be used for query execution.

```
PREFIX ORACLE_SEM_FS_NS: <http://oracle.com/semtech#dop=4>
PREFIX ex: <http://www.example.com/>
SELECT *
WHERE {?s ex:fname ?fname ;
          ex:lname ?lname ;
          ex:dob ?dob}
```

8.6.3.2 SPARQL_TO_SQL (SEM_MATCH) Options

You can pass `SPARQL_TO_SQL` options to the database server to influence the SQL generated for a SPARQL query by including a SPARQL `PREFIX` of the following form:

```
PREFIX ORACLE_SEM_SM_NS: <http://oracle.com/semtech#option>
```

The `option` in the above `PREFIX` reflects a `SPARQL_TO_SQL` option (or multiple options separated by commas) to be used during query execution.

The available options are detailed in Using the SEM_MATCH Table Function to Query RDF Data. Any valid keywords or keyword – value pairs listed as valid for the options argument of `SEM_MATCH` or `SEM_APIS.SPARQL_TO_SQL` can be used with this prefix.

The following example query uses the `ORACLE_SEM_SM_NS` prefix to specify that HASH join should be used to join all triple patterns in the query.

```
PREFIX ORACLE_SEM_SM_NS: <http://oracle.com/semtech#all_link_hash>
PREFIX ex: <http://www.example.org/>
SELECT *
WHERE {?s ex:fname ?fname ;
          ex:lname ?lname ;
          ex:dob ?dob}
```

8.6.4 Special Considerations for SPARQL Query Support

This section explains the special considerations for SPARQL Query Support.

Unbounded Property Path Queries

By default Oracle RDF Graph Adapter for Eclipse RDF4J limits the evaluation of the unbounded SPARQL property path operators `+` and `*` to at most 10 repetitions. This can be controlled with the `all_max_pp_depth(n) SPARQL_TO_SQL` option, where `n` is the maximum allowed number of repetitions when matching `+` or `*`. Specifying a value of zero results in unlimited maximum repetitions.

The following example uses `all_max_pp_depth(0)` for a fully unbounded search.

```
PREFIX ORACLE_SEM_SM_NS: <http://oracle.com/semtech#all_max_pp_depth(0)>
PREFIX ex: <http://www.example.org/>
SELECT (COUNT(*) AS ?cnt)
WHERE {ex:a ex:p1* ?y}
```

SPARQL Dataset Specification

The adapter for Eclipse RDF4J does not allow dataset specification outside of the SPARQL query string. Dataset specification through the `setDataset()` method of `Operation` and its subinterfaces is not supported, and passing a Dataset object into the `evaluate` method of `SailConnection` is also not supported. Instead, use the `FROM` and `FROM NAMED` SPARQL clauses to specify the query dataset in the SPARQL query string itself.

Query Timeout

Query timeout through the `setMaxExecutionTime` method on `Operation` and its subinterfaces is not supported.

Long RDF Literals

Large RDF literal values greater than 4000 bytes in length are not supported by some SPARQL query functions. See Special Considerations When Using SEM_MATCH for more information.

8.7 SPARQL Update Execution Model

This section explains the SPARQL Update Execution Model for Oracle RDF Graph Adapter for Eclipse RDF4J.

The adapter for Eclipse RDF4J implements SPARQL update operations by executing a sequence of SQL DML statements. By default, the logic to manage these SQL executions runs on the client, and JDBC is used to submit the SQL statements to the database server. However, setting the system property `oracle.rdf4j.adapter.sparqlInClient` to the value `F` will instead use a single invocation of the `SEM_APIS.UPDATE_RDF_GRAPH` stored procedure on the database server to execute the SPARQL update operation. Using `SEM_APIS.UPDATE_RDF_GRAPH` reduces database roundtrips and may result in better performance when there is a high latency between the client and database server. You can execute a SPARQL update operation by getting an `Update` object from the `prepareUpdate` function of an instance of `OracleSailRepositoryConnection`.

> **Note:**
>
> You must have an `OracleSailRepositoryConnection` instance. A plain `SailRepository` instance created from an `OracleSailStore` will not run the update properly.

The following example illustrates how to update an Oracle RDF graph through the RDF4J API:

```
String updString =
   "PREFIX people: <http://www.example.org/people/>\n"+
   "PREFIX    ont: <http://www.example.org/ontology/>\n"+
   "INSERT DATA { GRAPH <urn:g1> { \n"+
   "                 people:Sue a ont:Person; \n"+
   "                              ont:name \"Sue\" . } }";
   Update upd = conn.prepareUpdate(QueryLanguage.SPARQL, updString);
   upd.execute();
```

- Transaction Management for SPARQL Update
- Additions to the SPARQL Syntax to Support Other Features
- Special Considerations for SPARQL Update Support

8.7.1 Transaction Management for SPARQL Update

SPARQL update operations executed through the RDF4J API follow standard RDF4J transaction management conventions. SPARQL updates are committed automatically by default. However, if an explicit transaction is started on the `SailRepositoryConnection` with `begin`, then subsequent SPARQL update operations will not be committed until the active transaction is explicitly committed with `commit`. Any uncommitted update operations can be rolled back with `rollback`.

8.7.2 Additions to the SPARQL Syntax to Support Other Features

Just as it does with SPARQL queries, Oracle RDF Graph Adapter for Eclipse RDF4J allows you to pass in options for SPARQL update execution. It implements these capabilities by overloading the SPARQL namespace prefix syntax by using Oracle-specific namespaces that contain `SEM_APIS.UPDATE_RDF_GRAPH` options. These options apply to both client-based SPARQL Update execution and `SEM_APIS.UPDATE_RDF_GRAPH`-based execution.

- UPDATE_RDF_GRAPH Options
- UPDATE_RDF_GRAPH Match Options

8.7.2.1 UPDATE_RDF_GRAPH Options

You can pass options to SPARQL Update execution by including a `PREFIX` declaration with the following form:

```
PREFIX ORACLE_SEM_UM_NS: <http://oracle.com/semtech#option>
```

The `option` in the above `PREFIX` reflects an `UPDATE_RDF_GRAPH` option (or multiple options separated by commas) to be used during update execution.

See SEM_APIS.UPDATE_RDF_GRAPH for more information on available options. Any valid keywords or keyword – value pairs listed as valid for the options argument of `UPDATE_RDF_GRAPH` can be used with this `PREFIX`.

The following example query uses the `ORACLE_SEM_UM_NS` prefix to specify a degree of parallelism of 2 for the update.

```
PREFIX ORACLE_SEM_UM_NS: <http://oracle.com/semtech#parallel(2)>
PREFIX ex: <http://www.example.org/>
INSERT {GRAPH ex:g1 {ex:a ex:reachable ?y}}
WHERE {ex:a ex:p1* ?y}
```

8.7.2.2 UPDATE_RDF_GRAPH Match Options

You can pass match options to SPARQL Update execution by including a PREFIX declaration with the following form:

```
PREFIX ORACLE_SEM_SM_NS: <http://oracle.com/semtech#option>
```

The option reflects an UPDATE_RDF_GRAPH match option (or multiple match options separated by commas) to be used during SPARQL update execution.

The available options are detailed in SEM_APIS.UPDATE_RDF_GRAPH. Any valid keywords or keyword – value pairs listed as valid for the match_options argument of UPDATE_RDF_GRAPH can be used with this PREFIX.

The following example uses the ORACLE_SEM_SM_NS prefix to specify a maximum unbounded property path depth of 5.

```
PREFIX ORACLE_SEM_SM_NS: <http://oracle.com/semtech#all_max_pp_depth(5)>
PREFIX ex: <http://www.example.org/>
INSERT {GRAPH ex:g1 {ex:a ex:reachable ?y}}
WHERE {ex:a ex:p1* ?y}
```

8.7.3 Special Considerations for SPARQL Update Support

Unbounded Property Paths in Update Operations

As mentioned in the previous section, Oracle RDF Graph Adapter for Eclipse RDF4J limits the evaluation of the unbounded SPARQL property path operators + and * to at most 10 repetitions. This default setting will affect SPARQL update operations that use property paths in the WHERE clause. The max repetition setting can be controlled with the all_max_pp_depth(n) option, where n is the maximum allowed number of repetitions when matching + or *. Specifying a value of zero results in unlimited maximum repetitions.

The following example uses all_max_pp_depth(0) as a match option for a fully unbounded search during SPARQL Update execution.

```
PREFIX ORACLE_SEM_SM_NS: <http://oracle.com/semtech#all_max_pp_depth(0)>
PREFIX ex: <http://www.example.org/>
INSERT { GRAPH ex:g1 { ex:a ex:reachable ?y}}
WHERE { ex:a ex:p1* ?y}
```

SPARQL Dataset Specification

Oracle RDF Graph Adapter for Eclipse RDF4J does not allow dataset specification outside of the SPARQL update string. Dataset specification through the setDataset method of Operation and its subinterfaces is not supported. Instead, use the WITH, USING and USING NAMED SPARQL clauses to specify the dataset in the SPARQL update string itself.

Bind Values

Bind values are not supported for SPARQL update operations.

Long RDF Literals

As noted in the previous section, large RDF literal values greater than 4000 bytes in length are not supported by some SPARQL query functions. This limitation will affect SPARQL update operations using any of these functions on long literal data. See Special Considerations When Using SEM_MATCH for more information.

Update Timeout

Update timeout through the `setMaxExecutionTime` method on `Operation` and its subinterfaces is not supported.

8.8 Efficiently Loading RDF Data

The Oracle RDF Graph Adapter for Eclipse RDF4J provides additional or improved Java methods for efficiently loading a large amount of RDF data from files or collections.

Bulk Loading of RDF Data

The bulk loading capability of the adapter involves the following two steps:

1. Loading RDF data from a file or collection of statements to a staging table.
2. Loading RDF data from the staging table to the RDF storage tables.

The `OracleBulkUpdateHandler` class in the adapter provides methods that allow two different pathways for implementing a bulk load:

1. `addInBulk`: These methods allow performing both the steps mentioned in Bulk Loading of RDF Data with a single invocation. This pathway is better when you have only a single file or collection to load from.
2. `prepareBulk` and `completeBulk`: You can use one or more invocations of `prepareBulk`. Each call implements the step 1 of Bulk Loading of RDF Data.
   Later, a single invocation of `completeBulk` can be used to perform step 2 of Bulk Loading of RDF Data to load staging table data obtained from those multiple `prepareBulk` calls. This pathway works better when there are multiple files to load from.

In addition, the `OracleSailRepositoryConnection` class in the adapter provides bulk loading implementation for the following method in `SailRepositoryConnection` class: .

```
public void add(InputStream in,
                     String baseURI,
                     RDFFormat dataFormat,
                     Resource... contexts)
```

Bulk loading from compressed file is supported as well, but currently limited to gzip files only.

See Also:

ORARDFLDR Utility for Bulk Loading RDF Data

8.9 Validating RDF Data with SHACL Constraints

This section explains how to validate RDF graphs with SHACL (Shapes Constraint Language) constraints using Oracle RDF Graph Adapter for Eclipse RDF4J.

SHACL is a W3C standard for specifying constraints for RDF graphs. For example, SHACL allows you to specify that all instances of the `ex:Person` class must have a value for the `ex:name` property. SHACL defines an RDF vocabulary that allows you to specify constraints. An RDF graph that contains SHACL constraints is referred to as a shapes graph, and the RDF graph to be validated is referred to as a data graph.

RDF4J supports SHACL through its `ShaclSail` class. The SHACL engine in RDF4J validates a graph when changes to the graph are committed through RDF4J's transaction mechanism. See the RDF4J Documentation for general information about RDF4J's SHACL support.

When using `ShaclSail` with Oracle Adapter for Eclipse RDF4J through the `OracleShaclSail` class, a full SHACL validation runs when a transaction is committed. Incremental validation targeted at a subset of data is not supported. You should therefore avoid committing many small changes through an `OracleShaclSail` object, and use `OracleShaclSail` for large, bulk validation operations. The RDF4J SHACL engine sends a series of SPARQL queries to Oracle Database to check for constraint violations. A typical workflow for bulk validation is shown below:

1. Create an `OracleSailStore` object for an RDF data graph stored in Oracle Database.
2. Create an `OracleShaclSail` object that wraps the `OracleSailStore`.
3. Create a `SailRepository` from the `OracleShaclSail` object.
4. Begin a transaction on the `SailRepository`.
5. Add a shapes graph to the `RDF4J.SHACL_SHAPE_GRAPH` context as a part of the transaction (this shapes graph can be loaded from a variety of sources).
6. Commit the transaction.
7. A `RepositoryException` will be raised if a constraint is violated.
8. Check the validation report if an exception was raised.

These steps are illustrated in the following code fragment:

```
// Get an OracleSailStore instance for the stored data graph to validate
OraclePool op = new OraclePool(jdbcURL, user, password);
NotifyingSail store = new OracleSailStore(op, "DATA_GRAPH", "SCOTT", "NET1");

// Create an OracleShaclSail on top of the underlying OracleSailStore
ShaclSail shaclSail = new OracleShaclSail(store);
SailRepository sailRepository = new SailRepository(shaclSail);
sailRepository.init();

// Get a connection from the repository, start a transaction,
// load a shapes graph and commit the transaction to validate the data graph
try (SailRepositoryConnection conn = sailRepository.getConnection()) {
  conn.begin();
  // clear any existing shapes graph
  conn.clear(RDF4J.SHACL_SHAPE_GRAPH);
  // Add current shapes graph
  // Every person must have a name property
  StringReader shaclRules = new StringReader(
    String.join(
      "\n", "",
      "@prefix ex: <http://oracle.example.com/ns#> .",
      "@prefix sh: <http://www.w3.org/ns/shacl#> .",
      "@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .",
      "@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .",
      "@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .",

      "ex:MinCountShape",
      "  a sh:NodeShape ;",
      "  sh:targetClass ex:Person ;",
```

```
        "  sh:property [",
        "    sh:path ex:name ;",
        "    sh:minCount 1 ;",
        "  ] ."
      )
    );

    try {
      conn.add(shaclRules, null, RDFFormat.TURTLE, RDF4J.SHACL_SHAPE_GRAPH);
    }
    catch (IOException e) {
      e.printStackTrace();
    }

    // Commit transaction to validate the data graph with the current shapes
graph
    try {
      conn.commit();
    }
    catch (RepositoryException e) {
      Throwable cause = e.getCause();
      if (cause instanceof ValidationException) {
        Model validationReportModel = ((ValidationException)
cause).validationReportAsModel();

        WriterConfig writerConfig = new WriterConfig()
          .set(BasicWriterSettings.INLINE_BLANK_NODES, true)
          .set(BasicWriterSettings.XSD_STRING_TO_PLAIN_LITERAL, true)
          .set(BasicWriterSettings.PRETTY_PRINT, true);

        Rio.write(validationReportModel, System.out, RDFFormat.TURTLE,
writerConfig);
      }
      else {
        e.printStackTrace();
      }
    }
}
```

- SHACL Features Supported by Oracle Adapter for Eclipse RDF4J
  This section lists the SHACL core constraints supported by Oracle Adapter for Eclipse RDF4J.
- Restrictions on the use of RDF4J SHACL Features
  This section describes the restrictions on the use of RDF4J SHACL features.

8.9.1 SHACL Features Supported by Oracle Adapter for Eclipse RDF4J

This section lists the SHACL core constraints supported by Oracle Adapter for Eclipse RDF4J.

- `http://www.w3.org/ns/shacl#alternativePath`
- `http://www.w3.org/ns/shacl#class`
- `http://www.w3.org/ns/shacl#datatype`
- `http://www.w3.org/ns/shacl#deactivated`

- `http://www.w3.org/ns/shacl#flags`
- `http://www.w3.org/ns/shacl#hasValue`
- `http://www.w3.org/ns/shacl#in`
- `http://www.w3.org/ns/shacl#inversePath`
- `http://www.w3.org/ns/shacl#languageIn`
- `http://www.w3.org/ns/shacl#maxCount`
- `http://www.w3.org/ns/shacl#maxExclusive`
- `http://www.w3.org/ns/shacl#maxInclusive`
- `http://www.w3.org/ns/shacl#maxLength`
- `http://www.w3.org/ns/shacl#minCount`
- `http://www.w3.org/ns/shacl#minExclusive`
- `http://www.w3.org/ns/shacl#minInclusive`
- `http://www.w3.org/ns/shacl#minLength`
- `http://www.w3.org/ns/shacl#node`
- `http://www.w3.org/ns/shacl#nodeKind`
- `http://www.w3.org/ns/shacl#path`
- `http://www.w3.org/ns/shacl#pattern`
- `http://www.w3.org/ns/shacl#property`
- `http://www.w3.org/ns/shacl#severity`
- `http://www.w3.org/ns/shacl#target`
- `http://www.w3.org/ns/shacl#targetClass`
- `http://www.w3.org/ns/shacl#targetNode`
- `http://www.w3.org/ns/shacl#targetObjectsOf`
- `http://www.w3.org/ns/shacl#targetSubjectsOf`
- `http://www.w3.org/ns/shacl#uniqueLang`

The following features are not enabled by default but can be enabled by setting the system property `oracle.rdf4j.adapter.restrictShaclFeatures` to the value F. These features should only be used with smaller RDF datasets because the RDF4J SHACL engine's implementation reads the set of target nodes into the client program's memory when evaluating constraints with these features.

- `http://www.w3.org/ns/shacl#and`
- `http://www.w3.org/ns/shacl#not`
- `http://www.w3.org/ns/shacl#or`
- `http://www.w3.org/ns/shacl#qualifiedMaxCount`
- `http://www.w3.org/ns/shacl#qualifiedMinCount`
- `http://www.w3.org/ns/shacl#qualifiedValueShape`
- `http://www.w3.org/ns/shacl#sparql`

8.9.2 Restrictions on the use of RDF4J SHACL Features

This section describes the restrictions on the use of RDF4J SHACL features.

Oracle Adapter for Eclipse RDF4J only supports SHACL Shapes stored in the reserved named graph (context) `http://rdf4j.org/schema/rdf4j#SHACLShapeGraph`. In general, `ShaclSail` allows you to load shapes graphs from arbitrary named graphs identified with the `setShapesGraph` method. However, `setShapesGraph` is not supported by Oracle Adapter for Eclipse RDF4J.

8.10 ORARDFLDR Utility for Bulk Loading RDF Data

This section describes using the ORARDFLDR utility program for bulk loading RDF data serialized in various standard formats such as `RDF/XML`, `N-Triples`, `Turtle`, `JSON-LD`, and so on.

The Java class `oracle.rdf4j.adapter.utils.RDFLoader` is included with Oracle Adapter for Eclipse RDF4J. It uses Eclipse RDF4J's RDF parsers in combination with Oracle Database's scalable bulk loading capabilities to parse and load RDF files into Oracle Database.

This utility program loads all files in a directory into an RDF graph in Oracle Database. It supports several RDF serializations like `RDF/XML`, `Turtle`, `N-Triple`, `N-Quads`, `Trig`, and `JSON-LD`. Files compressed with `gzip` can be directly loaded without uncompressing the `gzip` file. In addition, Unicode characters, special characters, and long literals (`CLOB`s) are handled automatically.

Running ORARDFLDR Utility Program

The following describes the commands to execute ORARDFLDR. The commands assume that all Oracle support for Eclipse RDF4J jars and required supporting jars are located in `/tmp/oracle_adapter/jar` as explained previously in Setup and Configuration for Using Oracle RDF Graph Adapter for Eclipse RDF4J.

- **Usage:**

  ```
  java -cp /tmp/oracle_adapter/jar/'*' oracle.rdf4j.adapter.utils.RDFLoader
  <command_line_arguments>
  ```

- **For help details:**

  ```
  java -cp /tmp/oracle_adapter/jar/'*' oracle.rdf4j.adapter.utils.RDFLoader --help
  ```

For convenience, a shell script in the `bin` directory can also be executed. Proper set up of the `CLASSPATH` environment variable is required for execution of this shell script. The following describes the prerequisites and the commands to use this script:

Prerequisite: Set the `CLASSPATH` environment variable and ensure the `/bin` directory of the kit is in your Unix `PATH` environment variable.

- **Setup (assuming C shell command line interface):**

  ```
  export CLASSPATH=/tmp/oracle_adapter/jar/'*'
  export PATH=/tmp/oracle_adapter/bin:$PATH
  ```

- **Usage:**

  ```
  orardfldr <command_line_arguments>
  ```

- **For help details:**

```
orardfldr --help
```

8.11 Best Practices for Oracle RDF Graph Adapter for Eclipse RDF4J

This section explains the performance best practices for Oracle RDF Graph Adapter for Eclipse RDF4J.

Closing Resources

Application programmers should take care to avoid resource leaks. For Oracle RDF Graph Adapter for Eclipse RDF4J, the two most important types of resource leaks to prevent are JDBC connection leaks and database cursor leaks.

Preventing JDBC Connection Leaks

A new JDBC connection is obtained from the OraclePool every time you call `getConnection` on an `OracleRepository` or `OracleSailStore` to create an `OracleSailConnection` or `OracleSailRepositoryConnection` object. You must ensure that these JDBC connections are returned to the OraclePool by explicitly calling the `close` method on the `OracleSailConnection` or `OracleSailRepositoryConnection` objects that you create.

Preventing Database Cursor Leaks

Several RDF4J API calls return an Iterator. When using the adapter for Eclipse RDF4J, many of these iterators have underlying JDBC ResultSets that are opened when the iterator is created and therefore must be closed to prevent database cursor leaks.

Oracle's iterators can be closed in two ways:

1. By creating them in `try-with-resources` statements and relying on Java `Autoclosable` to close the iterator.

```
String queryString =
   "PREFIX ex: <http://example.org/ontology/>\n"+
   "SELECT * WHERE {?x ex:name ?y}\n" +
   "ORDER BY ASC(STR(?y)) LIMIT 1 ";

TupleQuery tupleQuery = conn.prepareTupleQuery(QueryLanguage.SPARQL,
queryString);

try (TupleQueryResult result = tupleQuery.evaluate()) {
  while (result.hasNext()) {
    BindingSet bindingSet = result.next();
    System.out.println("value of x: " + bindingSet.getValue("x"));
    System.out..println("value of y: " + bindingSet.getValue("y"))
  }
}
```

2. By explicitly calling the `close` method on the iterator.

```
String queryString =
  "PREFIX ex: <http://example.org/ontology/>\n"+
```

```
  "SELECT * WHERE {?x ex:name ?y}\n" +
  "ORDER BY ASC(STR(?y)) LIMIT 1 ";
TupleQuery tupleQuery = conn.prepareTupleQuery(QueryLanguage.SPARQL,
queryString);
TupleQueryResult result = tupleQuery.evaluate();
try {
  while (result.hasNext()) {
    BindingSet bindingSet = result.next();
    System.out.println("value of x: " +
bindingSet.getValue("x"));
    System.out..println("value of y: " + bindingSet.getValue("y"))
  }
}
finally {
  result.close();
}
```

Gathering Statistics

It is strongly recommended that you analyze the application table, RDF graph, and inferred graph in case it exists before performing inference and after loading a significant amount of RDF data into the database. Performing the analysis operations causes statistics to be gathered, which will help the Oracle optimizer select efficient execution plans when answering queries.

To gather relevant statistics, you can use the following methods in the `OracleSailConnection`:

- `OracleSailConnection.analyze`
- `OracleSailConnection.analyzeApplicationTable`

For information about these methods, including their parameters, see the RDF Graph Support for Eclipse RDF4J Javadoc.

JDBC Bind Values

It is strongly recommended that you use JDBC bind values whenever you execute a series of SPARQL queries that differ only in constant values. Using bind values saves significant query compilation overhead and can lead to much higher throughput for your query workload.

For more information about JDBC bind values, see Using JDBC BIND Values and Example 13: Using JDBC Bind Values.

8.12 Blank Nodes Support in Oracle RDF Graph Adapter for Eclipse RDF4J

In a SPARQL query, a blank node that is not wrapped inside < and > is treated as a variable when the query is executed through the support for the adapter for Eclipse RDF4J. This matches the SPARQL standard semantics.

However, a blank node that is wrapped inside < and > is treated as a constant when the query is executed, and the support for Eclipse RDF4J adds a proper prefix to the blank node label as required by the underlying data modeling. Do not use blank nodes for the `CONTEXT` column in the application table, because blank nodes in named graphs from two different RDF graphs will be treated as the same resource if they have the same label. This is not the case for blank nodes in triples, where they are stored separately if coming from different RDF graphs.

The blank node when stored in Oracle database is embedded with a prefix based on the RDF graph `ID` and graph name. Therefore, a conversion is needed between blank nodes used in RDF4J API's and Oracle Database. This can be done using the following methods:

- `OracleUtils.addOracleBNodePrefix`
- `OracleUtils.removeOracleBNodePrefix`

8.13 Unsupported Features in Oracle RDF Graph Adapter for Eclipse RDF4J

The unsupported features in the current version of Oracle RDF Graph Adapter for Eclipse RDF4J are discussed in this section.

The following features of Oracle RDF Graph are not supported in this version of the adapter for Eclipse RDF4J:

- RDF View graphs
- Native Unicode Storage (available in Oracle Database version 21c and later)
- Managing RDF graphs in Oracle Autonomous Database

The following features of the Eclipse RDF4J API are not supported in this version of the adapter for Eclipse RDF4J:

- SPARQL Dataset specification using the `setDataset` method of `Operation` and its subinterfaces is not supported. The dataset should be specified in the SPARQL query or update string itself.
- Specifying Query and Update timeout through the `setMaxExecutionTime` method on `Operation` and its subinterfaces is not supported.
- A `TupleExpr` that does not implement `OracleTuple` cannot be passed to the `evaluate` method in `OracleSailConnection`.
- An Update object created from a `RepositoryConnection` implementation other than `OracleSailRepositoryConnection` cannot be executed against Oracle RDF

8.14 Example Queries Using Oracle RDF Graph Adapter for Eclipse RDF4J

This section includes the example queries for using Oracle RDF Graph Adapter for Eclipse RDF4J.

To run these examples, ensure that all the supporting libraries mentioned in Supporting libraries for using adapter with Java code are included in the `CLASSPATH` definition.

To run a query, you must execute the following actions:

1. Include the example code in a Java source file.
2. Define a `CLASSPATH` environment variable named `CP` to include the relevant jar files. For example, it may be defined as follows:

```
setenv CP .:ojdbc8.jar:ucp.jar:oracle-rdf4j-
adapter-4.3.14-20250106.jar:sdordf-23.6.0-20241122.jar:
sdordf-client-23.6.0-20241122.jar:sdoutl-23.6.0-20241122.jar:log4j-
api-2.24.2.jar:log4j-core-2.24.2.jar:
```

```
log4j-slf4j-impl-2.24.2.jar:slf4j-api-1.7.36.jar:eclipse-rdf4j-4.3.14-
onejar.jar:commons-io-2.14.0.jar
```

> **Note:**
>
> The preceding `setenv` command assumes that the jar files are located in the current directory. You may need to alter the command to indicate the location of these jar files in your environment.

3. Compile the Java source file. For example, to compile the source file `Test.java`, run the following command:

```
javac -classpath $CP Test.java
```

4. Run the compiled file on an RDF graph named `TestModel` in an existing schema-private network whose owner is SCOTT and name is NET1 by executing the following command:

```
java -classpath $CP Test jdbc:oracle:thin:@localhost:1521:orcl scott
<password-for-scott> TestModel scott net1
```


8.14.1 Example 1: Basic Operations

Example 8-5 shows the `BasicOper.java` file, which performs some basic operations such as add and remove statements.

Example 8-5 Basic Operations

```
import java.io.IOException;
import java.io.PrintStream;
```

```
import java.sql.SQLException;
import oracle.rdf4j.adapter.OraclePool;
import oracle.rdf4j.adapter.OracleRepository;
import oracle.rdf4j.adapter.OracleSailConnection;
import oracle.rdf4j.adapter.OracleSailStore;
import oracle.rdf4j.adapter.exception.ConnectionSetupException;
import oracle.rdf4j.adapter.utils.OracleUtils;
import org.eclipse.rdf4j.common.iteration.CloseableIteration;
import org.eclipse.rdf4j.model.IRI;
import org.eclipse.rdf4j.model.Statement;
import org.eclipse.rdf4j.model.ValueFactory;
import org.eclipse.rdf4j.repository.Repository;
import org.eclipse.rdf4j.sail.SailException;

public class BasicOper {
  public static void main(String[] args) throws ConnectionSetupException,
SQLException, IOException {
    PrintStream psOut = System.out;
    String jdbcUrl = args[0];
    String user = args[1];
    String password = args[2];
    String model = args[3];
    String networkOwner = (args.length > 5) ? args[4] : null;
    String networkName = (args.length > 5) ? args[5] : null;
    OraclePool op = null;
    OracleSailStore store = null;
    Repository sr = null;
    OracleSailConnection conn = null;

    try {
      op = new OraclePool(jdbcUrl, user, password);
      store = new OracleSailStore(op, model, networkOwner, networkName);
      sr = new OracleRepository(store);

      ValueFactory f = sr.getValueFactory();
      conn = store.getConnection();

      // create some resources and literals to make statements out of
      IRI p = f.createIRI("http://p");
      IRI domain = f.createIRI("http://www.w3.org/2000/01/rdf-schema#domain");
      IRI cls = f.createIRI("http://cls");
      IRI a = f.createIRI("http://a");
      IRI b = f.createIRI("http://b");
      IRI ng1 = f.createIRI("http://ng1");

      conn.addStatement(p, domain, cls);
      conn.addStatement(p, domain, cls, ng1);
      conn.addStatement(a, p, b, ng1);
      psOut.println("size for given contexts " + ng1 + ": " + conn.size(ng1));

      // returns OracleStatements
      CloseableIteration < ?extends Statement, SailException > it;
      int cnt;

      // retrieves all statements that appear in the repository(regardless of
context)
```

```
      cnt = 0;
      it = conn.getStatements(null, null, null, false);
      while (it.hasNext()) {
        Statement stmt = it.next();
        psOut.println("getStatements: stmt#" + (++cnt) + ":" +
stmt.toString());
      }
      it.close();
      conn.removeStatements(null, null, null, ng1);
      psOut.println("size of context " + ng1 + ":" + conn.size(ng1));
      conn.removeAll();
      psOut.println("size of store: " + conn.size());
    }

    finally {
      if (conn != null && conn.isOpen()) {
        conn.close();
      }
      if (op != null && op.getOracleDB() != null)

      OracleUtils.dropSemanticModelAndTables(op.getOracleDB(), model, null,
null, networkOwner, networkName);
      if (sr != null) sr.shutDown();
      if (store != null) store.shutDown();
      if (op != null) op.close();
    }
  }
}
```

To compile this example, execute the following command:

```
javac -classpath $CP BasicOper.java
```

To run this example for an existing schema-private network whose owner is SCOTT and name is NET1, execute the following command:

```
java -classpath $CP BasicOper jdbc:oracle:thin:@localhost:1521:ORCL scott
<password-for-scott> TestModel scott net1
```

The expected output of the `java` command might appear as follows:

```
size for given contexts http://ng1: 2
getStatements: stmt#1: (http://a, http://p, http://b) [http://ng1]
getStatements: stmt#2: (http://p, http://www.w3.org/2000/01/rdf-
schema#domain, http://cls) [http://ng1]
getStatements: stmt#3: (http://p, http://www.w3.org/2000/01/rdf-
schema#domain, http://cls) [null]
size of context http://ng1:0
size of store: 0
```

8.14.2 Example 2: Add a Data File in TRIG Format

Add a Data File in TRIG Format shows the `LoadFile.java` file, which demonstrates how to load a file in TRIG format.

Example 8-6   Add a Data File in TRIG Format

```
import java.io. * ;
import java.sql.SQLException;
import org.eclipse.rdf4j.repository.Repository;
import org.eclipse.rdf4j.repository.RepositoryConnection;
import org.eclipse.rdf4j.repository.RepositoryException;
import org.eclipse.rdf4j.rio.RDFParseException;
import org.eclipse.rdf4j.sail.SailException;
import org.eclipse.rdf4j.rio.RDFFormat;
import oracle.rdf4j.adapter.OraclePool;
import oracle.rdf4j.adapter.OracleRepository;
import oracle.rdf4j.adapter.OracleSailConnection;
import oracle.rdf4j.adapter.OracleSailStore;
import oracle.rdf4j.adapter.exception.ConnectionSetupException;
import oracle.rdf4j.adapter.utils.OracleUtils;

public class LoadFile {
  public static void main(String[] args) throws ConnectionSetupException,
    SQLException, SailException, RDFParseException, RepositoryException,
    IOException {

      PrintStream psOut = System.out;
      String jdbcUrl = args[0];
      String user = args[1];
      String password = args[2];
      String model = args[3];
      String trigFile = args[4];
      String networkOwner = (args.length > 6) ? args[5] : null;
      String networkName = (args.length > 6) ? args[6] : null;


      OraclePool op = null;
      OracleSailStore store = null;
      Repository sr = null;
      RepositoryConnection repConn = null;

      try {
        op = new OraclePool(jdbcUrl, user, password);
        store = new OracleSailStore(op, model, networkOwner, networkName);
        sr = new OracleRepository(store);
        repConn = sr.getConnection();
        psOut.println("testBulkLoad: start: before-load Size=" +
repConn.size());
        repConn.add(new File(trigFile), "http://my.com/", RDFFormat.TRIG);
        repConn.commit();
        psOut.println("size " + Long.toString(repConn.size()));
      }
      finally {
        if (repConn != null) {
```

```
          repConn.close();
        }
        if (op != null)
OracleUtils.dropSemanticModelAndTables(op.getOracleDB(), model, null, null,
networkOwner, networkName);
        if (sr != null) sr.shutDown();
        if (store != null) store.shutDown();
        if (op != null) op.close();
      }
  }
}
```

For running this example, assume that a sample TRIG data file named `test.trig` was created as:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix swp: <http://www.w3.org/2004/03/trix/swp-1/>.
@prefix dc: <http://purl.org/dc/elements/1.1/>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
@prefix ex: <http://example.org/>.
@prefix : <http://example.org/>.
# default graph
{
  <http://example.org/bob>   dc:publisher "Bob Hacker".
  <http://example.org/alice> dc:publisher "Alice Hacker".
}
:bob{
    _:a foaf:mbox <mailto:bob@oldcorp.example.org>.
    }
:alice{
      _:a foaf:name "Alice".
      _:a foaf:mbox <mailto:alice@work.example.org>.
      }
:jack {
      _:a foaf:name "Jack".
      _:a foaf:mbox <mailto:jack@oracle.example.org>.
      }
```

To compile this example, execute the following command:

```
javac -classpath $CP LoadFile.java
```

To run this example for an existing schema-private network whose owner is SCOTT and name is NET1, execute the following command:

```
java -classpath $CP LoadFile jdbc:oracle:thin:@localhost:1521:ORCL scott
<password>  TestModel ./test.trig scott net1
```

The expected output of the `java` command might appear as follows:

```
testBulkLoad: start: before-load Size=0
size 7
```

8.14.3 Example 3: Simple Query

Example 3: Simple Query shows the `SimpleQuery.java` file, which demonstrates how to perform a simple query.

Example 8-7 Simple Query

```
import java.io.IOException;
import java.io.PrintStream;
import java.sql.SQLException;
import oracle.rdf4j.adapter.OraclePool;
import oracle.rdf4j.adapter.OracleRepository;
import oracle.rdf4j.adapter.OracleSailStore;
import oracle.rdf4j.adapter.exception.ConnectionSetupException;
import oracle.rdf4j.adapter.utils.OracleUtils;
import org.eclipse.rdf4j.model.IRI;
import org.eclipse.rdf4j.model.Literal;
import org.eclipse.rdf4j.model.ValueFactory;
import org.eclipse.rdf4j.model.vocabulary.RDF;
import org.eclipse.rdf4j.query.BindingSet;
import org.eclipse.rdf4j.query.QueryLanguage;
import org.eclipse.rdf4j.query.TupleQuery;
import org.eclipse.rdf4j.query.TupleQueryResult;
import org.eclipse.rdf4j.repository.Repository;
import org.eclipse.rdf4j.repository.RepositoryConnection;

public class SimpleQuery {
  public static void main(String[] args) throws ConnectionSetupException,
SQLException, IOException {
    PrintStream psOut = System.out;
    String jdbcUrl = args[0];
    String user = args[1];
    String password = args[2];
    String model = args[3];
    String networkOwner = (args.length > 5) ? args[4] : null;
    String networkName = (args.length > 5) ? args[5] : null;


    OraclePool op = null;
    OracleSailStore store = null;
    Repository sr = null;
    RepositoryConnection conn = null;

    try {
      op = new OraclePool(jdbcUrl, user, password);
      store = new OracleSailStore(op, model, networkOwner, networkName);
      sr = new OracleRepository(store);

      ValueFactory f = sr.getValueFactory();
```

```
      conn = sr.getConnection();

      // create some resources and literals to make statements out of
      IRI alice = f.createIRI("http://example.org/people/alice");
      IRI name = f.createIRI("http://example.org/ontology/name");
      IRI person = f.createIRI("http://example.org/ontology/Person");
      Literal alicesName = f.createLiteral("Alice");

      conn.clear(); // to start from scratch
      conn.add(alice, RDF.TYPE, person);
      conn.add(alice, name, alicesName);
      conn.commit();

      //run a query against the repository
      String queryString =
        "PREFIX ex: <http://example.org/ontology/>\n" +
        "SELECT * WHERE {?x ex:name ?y}\n" +
        "ORDER BY ASC(STR(?y)) LIMIT 1 ";
      TupleQuery tupleQuery = conn.prepareTupleQuery(QueryLanguage.SPARQL,
queryString);

      try (TupleQueryResult result = tupleQuery.evaluate()) {
        while (result.hasNext()) {
          BindingSet bindingSet = result.next();
          psOut.println("value of x: " + bindingSet.getValue("x"));
          psOut.println("value of y: " + bindingSet.getValue("y"));
        }
      }
    }
    finally {
      if (conn != null && conn.isOpen()) {
        conn.clear();
        conn.close();
      }
      OracleUtils.dropSemanticModelAndTables(op.getOracleDB(), model, null,
null, networkOwner, networkName);
      sr.shutDown();
      store.shutDown();
      op.close();
    }
  }
}
```

To compile this example, execute the following command:

```
javac -classpath $CP SimpleQuery.java
```

To run this example for an existing schema-private network whose owner is SCOTT and name is NET1, execute the following command:

```
java -classpath $CP SimpleQuery jdbc:oracle:thin:@localhost:1521:ORCL scott
<password-for-scott> TestModel scott net1
```

The expected output of the `java` command might appear as follows:

```
value of x: http://example.org/people/alice
value of y: "Alice"
```

8.14.4 Example 4: Simple Bulk Load

Example 8-8 shows the `SimpleBulkLoad.java` file, which demonstrates how to do a bulk load from NTriples data.

Example 8-8 Simple Bulk Load

```
import java.io. * ;
import java.sql.SQLException;
import org.eclipse.rdf4j.model.IRI;
import org.eclipse.rdf4j.model.ValueFactory;
import org.eclipse.rdf4j.model.Resource;
import org.eclipse.rdf4j.repository.RepositoryException;
import org.eclipse.rdf4j.rio.RDFParseException;
import org.eclipse.rdf4j.sail.SailException;
import org.eclipse.rdf4j.rio.RDFFormat;
import org.eclipse.rdf4j.repository.Repository;
import oracle.rdf4j.adapter.OraclePool;
import oracle.rdf4j.adapter.OracleRepository;
import oracle.rdf4j.adapter.OracleSailConnection;
import oracle.rdf4j.adapter.OracleSailStore;
import oracle.rdf4j.adapter.exception.ConnectionSetupException;
import oracle.rdf4j.adapter.utils.OracleUtils;

public class SimpleBulkLoad {
  public static void main(String[] args) throws ConnectionSetupException,
SQLException,
    SailException, RDFParseException, RepositoryException, IOException {
      PrintStream psOut = System.out;
      String jdbcUrl = args[0];
      String user = args[1];
      String password = args[2];
      String model = args[3];
      String filename = args[4]; // N-TRIPLES file
      String networkOwner = (args.length > 6) ? args[5] : null;
      String networkName = (args.length > 6) ? args[6] : null;


      OraclePool op = new OraclePool(jdbcUrl, user, password);
      OracleSailStore store = new OracleSailStore(op, model, networkOwner,
networkName);
      OracleSailConnection osc = store.getConnection();
      Repository sr = new OracleRepository(store);
      ValueFactory f = sr.getValueFactory();

      try {
        psOut.println("testBulkLoad: start");
```

```
        FileInputStream fis = new
        FileInputStream(filename);

        long loadBegin = System.currentTimeMillis();
        IRI ng1 = f.createIRI("http://QuadFromTriple");
        osc.getBulkUpdateHandler().addInBulk(
        fis, "http://abc",  // baseURI
        RDFFormat.NTRIPLES, // dataFormat
        null,               // tablespaceName
        50,                 // batchSize
        null,               // flags
        ng1                 // Resource... for contexts
        );

        long loadEnd = System.currentTimeMillis();
        long size_no_contexts = osc.size((Resource) null);
        long size_all_contexts = osc.size();

        psOut.println("testBulkLoad: " + (loadEnd - loadBegin) +
         "ms. Size:" + " NO_CONTEXTS=" + size_no_contexts + " ALL_CONTEXTS="
+ size_all_contexts);
        // cleanup
        osc.removeAll();
        psOut.println("size of store: " + osc.size());

      }
      finally {
        if (osc != null && osc.isOpen()) osc.close();
        if (op != null)
OracleUtils.dropSemanticModelAndTables(op.getOracleDB(), model, null, null,
networkOwner, networkName);
        if (sr != null) sr.shutDown();
        if (store != null) store.shutDown();
        if (op != null) op.close();
      }
  }
}
```

For running this example, assume that a sample ntriples data file named `test.ntriples` was created as:

```
<urn:JohnFrench> <urn:name> "John".
<urn:JohnFrench> <urn:speaks> "French".
<urn:JohnFrench> <urn:height> <urn:InchValue>.
<urn:InchValue> <urn:value> "63".
<urn:InchValue> <urn:unit> "inch".
<http://data.linkedmdb.org/movie/onto/genreNameChainElem1> <http://www.w3.org/
1999/02/22-rdf-syntax-ns#first> <http://data.linkedmdb.org/movie/genre>.
```

To compile this example, execute the following command:

```
javac -classpath $CP SimpleBulkLoad.java
```

To run this example for an existing schema-private network whose owner is SCOTT and name is NET1, execute the following command:

```
java -classpath $CP SimpleBulkLoad jdbc:oracle:thin:@localhost:1521:ORCL
scott <password> TestModel ./test.ntriples scott net1
```

The expected output of the `java` command might appear as follows:

```
testBulkLoad: start
testBulkLoad: 8222ms.
Size: NO_CONTEXTS=0 ALL_CONTEXTS=6
size of store: 0
```

8.14.5 Example 5: Bulk Load RDF/XML

Example 5: Bulk Load RDF/XML shows the `BulkLoadRDFXML.java` file, which demonstrates how to do a bulk load from RDF/XML file.

Example 8-9 Bulk Load RDF/XML

```
import java.io. * ;
import java.sql.SQLException;
import org.eclipse.rdf4j.model.Resource;
import org.eclipse.rdf4j.repository.Repository;
import org.eclipse.rdf4j.repository.RepositoryConnection;
import org.eclipse.rdf4j.repository.RepositoryException;
import org.eclipse.rdf4j.rio.RDFParseException;
import org.eclipse.rdf4j.sail.SailException;
import org.eclipse.rdf4j.rio.RDFFormat;
import oracle.rdf4j.adapter.OraclePool;
import oracle.rdf4j.adapter.OracleRepository;
import oracle.rdf4j.adapter.OracleSailConnection;
import oracle.rdf4j.adapter.OracleSailStore;
import oracle.rdf4j.adapter.exception.ConnectionSetupException;
import oracle.rdf4j.adapter.utils.OracleUtils;

public class BulkLoadRDFXML {
  public static void main(String[] args) throws
    ConnectionSetupException, SQLException, SailException,
    RDFParseException, RepositoryException, IOException {
      PrintStream psOut = System.out;
      String jdbcUrl = args[0];
      String user = args[1];
      String password = args[2];
      String model = args[3];
      String rdfxmlFile = args[4]; // RDF/XML-format data file
      String networkOwner = (args.length > 6) ? args[5] : null;
      String networkName = (args.length > 6) ? args[6] : null;


      OraclePool op = null;
      OracleSailStore store = null;
      Repository sr = null;
      OracleSailConnection conn = null;
```

```
      try {
        op = new OraclePool(jdbcUrl, user, password);
        store = new OracleSailStore(op, model, networkOwner, networkName);
        sr = new OracleRepository(store);
        conn = store.getConnection();

        FileInputStream fis = new FileInputStream(rdfxmlFile);
        psOut.println("testBulkLoad: start: before-load Size=" + conn.size());
        long loadBegin = System.currentTimeMillis();
        conn.getBulkUpdateHandler().addInBulk(
          fis,
          "http://abc",      // baseURI
          RDFFormat.RDFXML,  // dataFormat
          null,              // tablespaceName
          null,              // flags
          null,             //  StatusListener
          (Resource[]) null //  Resource...for contexts
        );

        long loadEnd = System.currentTimeMillis();
        psOut.println("testBulkLoad: " + (loadEnd - loadBegin) + "ms. Size="
+ conn.size() + "\n");
      }
      finally {
        if (conn != null && conn.isOpen()) {
          conn.close();
        }
        if (op != null)
OracleUtils.dropSemanticModelAndTables(op.getOracleDB(), model, null, null,
networkOwner, networkName);
        if (sr != null) sr.shutDown();
        if (store != null) store.shutDown();
        if (op != null) op.close();
      }
  }
}
```

For running this example, assume that a sample file named `RdfXmlData.rdfxml` was created as:

```
<?xml version="1.0"?>
<!DOCTYPE owl [
  <!ENTITY owl  "http://www.w3.org/2002/07/owl#" >
  <!ENTITY xsd  "http://www.w3.org/2001/XMLSchema#" >
]>
<rdf:RDF
  xmlns     = "http://a/b#" xml:base  = "http://a/b#" xmlns:my  =
"http://a/b#"
  xmlns:owl = "http://www.w3.org/2002/07/owl#"
  xmlns:rdf = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs= "http://www.w3.org/2000/01/rdf-schema#"
  xmlns:xsd = "http://www.w3.org/2001/XMLSchema#">
  <owl:Class rdf:ID="Color">
```

```
    <owl:oneOf rdf:parseType="Collection">
      <owl:Thing rdf:ID="Red"/>
      <owl:Thing rdf:ID="Blue"/>
    </owl:oneOf>
  </owl:Class>
</rdf:RDF>
```

To compile this example, execute the following command:

```
javac -classpath $CP BulkLoadRDFXML.java
```

To run this example for an existing schema-private network whose owner is SCOTT and name is NET1, execute the following command:

```
java -classpath $CP BulkLoadRDFXML jdbc:oracle:thin:@localhost:1521:ORCL
scott <password>  TestModel ./RdfXmlData.rdfxml scott net1
```

The expected output of the `java` command might appear as follows:

```
testBulkLoad: start: before-load Size=0
testBulkLoad: 6732ms. Size=8
```

8.14.6 Example 6: SPARQL Ask Query

Example 6: SPARQL Ask Query shows the `SparqlASK.java` file, which demonstrates how to perform a SPARQL ASK query.

Example 8-10   SPARQL Ask Query

```
import java.io.PrintStream;
import java.sql.SQLException;
import oracle.rdf4j.adapter.OraclePool;
import oracle.rdf4j.adapter.OracleRepository;
import oracle.rdf4j.adapter.OracleSailConnection;
import oracle.rdf4j.adapter.OracleSailRepositoryConnection;
import oracle.rdf4j.adapter.OracleSailStore;
import oracle.rdf4j.adapter.exception.ConnectionSetupException;
import oracle.rdf4j.adapter.utils.OracleUtils;
import org.eclipse.rdf4j.model.IRI;
import org.eclipse.rdf4j.model.ValueFactory;
import org.eclipse.rdf4j.model.vocabulary.RDFS;
import org.eclipse.rdf4j.query.BooleanQuery;
import org.eclipse.rdf4j.query.QueryLanguage;
import org.eclipse.rdf4j.repository.Repository;
import org.eclipse.rdf4j.repository.RepositoryConnection;

public class SparqlASK {
  public static void main(String[] args) throws ConnectionSetupException,
SQLException {
    PrintStream psOut = System.out;
    String jdbcUrl = args[0];
    String user = args[1];
    String password = args[2];
```

```
    String model = args[3];
    String networkOwner = (args.length > 5) ? args[4] : null;
    String networkName = (args.length > 5) ? args[5] : null;


    OraclePool op = null;
    OracleSailStore store = null;
    Repository sr = null;
    RepositoryConnection conn = null;

    try {
      op = new OraclePool(jdbcUrl, user, password);
      store = new OracleSailStore(op, model, networkOwner, networkName);
      sr = new OracleRepository(store);
      conn = sr.getConnection();
      OracleSailConnection osc =
        (OracleSailConnection)((OracleSailRepositoryConnection)
conn).getSailConnection();

      ValueFactory vf = sr.getValueFactory();
      IRI p = vf.createIRI("http://p");
      IRI cls = vf.createIRI("http://cls");

      conn.clear();
      conn.add(p, RDFS.DOMAIN, cls);
      conn.commit();

      osc.analyze();                 // analyze the semantic model
      osc.analyzeApplicationTable(); // and then the application table
      BooleanQuery tq = null;
      tq = conn.prepareBooleanQuery(QueryLanguage.SPARQL, "ASK { ?x ?p
<http://cls> }");
      boolean b = tq.evaluate();
      psOut.println("\nAnswer is " + Boolean.toString(b));
    }
    finally {
      if (conn != null && conn.isOpen()) {
        conn.clear();
        conn.close();
      }
      OracleUtils.dropSemanticModelAndTables(op.getOracleDB(), model, null,
null, networkOwner, networkName);
      sr.shutDown();
      store.shutDown();
      op.close();
    }
  }
}
```

To compile this example, execute the following command:

```
javac -classpath $CP SparqlASK.java
```

To run this example for an existing schema-private network whose owner is SCOTT and name is NET1, execute the following command:

```
java -classpath $CP SparqlASK jdbc:oracle:thin:@localhost:1521:ORCL scott
<password> TestModel scott net1
```

The expected output of the `java` command might appear as follows:

```
Answer is true
```

8.14.7 Example 7: SPARQL CONSTRUCT Query

Example 8-11 shows the `SparqlConstruct.java` file, which demonstrates how to perform a SPARQL CONSTRUCT query.

Example 8-11 SPARQL CONSTRUCT Query

```
import java.io.PrintStream;
import java.sql.SQLException;
import oracle.rdf4j.adapter.OraclePool;
import oracle.rdf4j.adapter.OracleRepository;
import oracle.rdf4j.adapter.OracleSailConnection;
import oracle.rdf4j.adapter.OracleSailRepositoryConnection;
import oracle.rdf4j.adapter.OracleSailStore;
import oracle.rdf4j.adapter.exception.ConnectionSetupException;
import oracle.rdf4j.adapter.utils.OracleUtils;
import org.eclipse.rdf4j.model.IRI;
import org.eclipse.rdf4j.model.Statement;
import org.eclipse.rdf4j.model.ValueFactory;
import org.eclipse.rdf4j.model.vocabulary.RDFS;
import org.eclipse.rdf4j.query.GraphQuery;
import org.eclipse.rdf4j.query.GraphQueryResult;
import org.eclipse.rdf4j.query.QueryLanguage;
import org.eclipse.rdf4j.repository.Repository;
import org.eclipse.rdf4j.repository.RepositoryConnection;

public class SparqlConstruct {
  public static void main(String[] args) throws ConnectionSetupException,
SQLException {
    PrintStream psOut = System.out;
    String jdbcUrl = args[0];
    String user = args[1];
    String password = args[2];
    String model = args[3];
    String networkOwner = (args.length > 5) ? args[4] : null;
    String networkName = (args.length > 5) ? args[5] : null;


    OraclePool op = null;
    OracleSailStore store = null;
    Repository sr = null;
    RepositoryConnection conn = null;

    try {
```

```
      op = new OraclePool(jdbcUrl, user, password);
      store = new OracleSailStore(op, model, networkOwner, networkName);
      sr = new OracleRepository(store);
      conn = sr.getConnection();

      ValueFactory vf = sr.getValueFactory();
      IRI p = vf.createIRI("http://p");
      IRI cls = vf.createIRI("http://cls");

      conn.clear();
      conn.add(p, RDFS.DOMAIN, cls);
      conn.commit();
      OracleSailConnection osc =
        (OracleSailConnection)((OracleSailRepositoryConnection)
conn).getSailConnection();
      osc.analyze();                 // analyze the RDF graph
      osc.analyzeApplicationTable(); // and then the application table

      GraphQuery tq = null;          // Construct Query
      tq = conn.prepareGraphQuery(QueryLanguage.SPARQL,
        "CONSTRUCT {?x <http://new_eq_p> ?o } WHERE { ?x ?p ?o }");
      psOut.println("Start construct query");

      try (GraphQueryResult result = tq.evaluate()) {
        while (result.hasNext()) {
          Statement stmt = (Statement) result.next();
          psOut.println(stmt.toString());
        }
      }
    }
    finally {
      if (conn != null && conn.isOpen()) {
        conn.clear();
        conn.close();
      }
      OracleUtils.dropSemanticModelAndTables(op.getOracleDB(), model, null,
null, networkOwner, networkName);
      sr.shutDown();
      store.shutDown();
      op.close();
    }
  }
}
```

To compile this example, execute the following command:

```
javac -classpath $CP SparqlConstruct.java
```

To run this example for an existing schema-private network whose owner is SCOTT and name is NET1, execute the following command:

```
java -classpath $CP SparqlConstruct jdbc:oracle:thin:@localhost:1521:ORCL
scott <password> TestModel scott net1
```

The expected output of the `java` command might appear as follows:

```
Start construct query
(http://p, http://new_eq_p, http://cls)
```

8.14.8 Example 8: Named Graph Query

Example 8-12 shows the `NamedGraph.java` file, which demonstrates how to perform a Named Graph query.

Example 8-12 Named Graph Query

```
import java.io.File;
import java.io.IOException;
import java.io.PrintStream;
import java.sql.SQLException;
import oracle.rdf4j.adapter.OraclePool;
import oracle.rdf4j.adapter.OracleRepository;
import oracle.rdf4j.adapter.OracleSailConnection;
import oracle.rdf4j.adapter.OracleSailRepositoryConnection;
import oracle.rdf4j.adapter.OracleSailStore;
import oracle.rdf4j.adapter.exception.ConnectionSetupException;
import oracle.rdf4j.adapter.utils.OracleUtils;
import org.eclipse.rdf4j.query.BindingSet;
import org.eclipse.rdf4j.query.QueryLanguage;
import org.eclipse.rdf4j.query.TupleQuery;
import org.eclipse.rdf4j.query.TupleQueryResult;
import org.eclipse.rdf4j.repository.Repository;
import org.eclipse.rdf4j.repository.RepositoryConnection;
import org.eclipse.rdf4j.rio.RDFFormat;

public class NamedGraph {
  public static void main(String[] args) throws ConnectionSetupException,
SQLException, IOException {
    PrintStream psOut = System.out;
    String jdbcUrl = args[0];
    String user = args[1];
    String password = args[2];
    String model = args[3];
    String trigFile = args[4]; // TRIG-format data file
    String networkOwner = (args.length > 6) ? args[5] : null;
    String networkName = (args.length > 6) ? args[6] : null;


    OraclePool op = null;
    OracleSailStore store = null;
    Repository sr = null;
    RepositoryConnection conn = null;

    try {
      op = new OraclePool(jdbcUrl, user, password);
      store = new OracleSailStore(op, model, networkOwner, networkName);
      sr = new OracleRepository(store);
      conn = sr.getConnection();
```

```
      conn.begin();
      conn.clear();

      // load the data incrementally since it is very small file
      conn.add(new File(trigFile), "http://my.com/", RDFFormat.TRIG);
      conn.commit();

      OracleSailConnection osc = (OracleSailConnection)
((OracleSailRepositoryConnection) conn).getSailConnection();

      osc.analyze(); // analyze the RDF graph
      osc.analyzeApplicationTable(); // and then the application table
      TupleQuery tq = null;
      tq = conn.prepareTupleQuery(QueryLanguage.SPARQL,
             "PREFIX : <http://purl.org/dc/elements/1.1/>\n" +
             "SELECT ?g ?s ?p ?o\n" +
             "WHERE {?g :publisher ?o1 . GRAPH ?g {?s ?p ?o}}\n" +
             "ORDER BY ?g ?s ?p ?o");
      try (TupleQueryResult result = tq.evaluate()) {
        int idx = 0;
        while (result.hasNext()) {
          idx++;
          BindingSet bindingSet = result.next();
          psOut.print("\nsolution " + bindingSet.toString());
        }
        psOut.println("\ntotal # of solution " + Integer.toString(idx));
      }
    }
    finally {
      if (conn != null && conn.isOpen()) {
        conn.clear();
        conn.close();
      }
      OracleUtils.dropSemanticModelAndTables(op.getOracleDB(), model, null,
null,  networkOwner, networkName);
      sr.shutDown();
      store.shutDown();
      op.close();
    }
  }
}
```

For running this example, assume that the `test.trig` file in `TRIG` format has been created as follows:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix swp: <http://www.w3.org/2004/03/trix/swp-1/>.
@prefix dc: <http://purl.org/dc/elements/1.1/>.
@prefix foaf: <http://xmlns.com/foaf/0.1/>.
@prefix : <http://example.org/>.
# default graph
{
```

```
  :bobGraph    dc:publisher  "Bob Hacker" .
  :aliceGraph  dc:publisher  "Alice Hacker" .
}

:bobGraph {
  :bob foaf:mbox <mailto:bob@oldcorp.example.org> .
}

:aliceGraph {
  :alice foaf:name "Alice" .
  :alice foaf:mbox <mailto:alice@work.example.org> .
}

:jackGraph {
  :jack foaf:name "Jack" .
  :jack foaf:mbox <mailto:jack@oracle.example.org> .
}
```

To compile this example, execute the following command:

```
javac -classpath $CP NamedGraph.java
```

To run this example for an existing schema-private network whose owner is SCOTT and name is NET1, execute the following command:

```
java -classpath $CP NamedGraph jdbc:oracle:thin:@localhost:1521:ORCL scott
<password> TestModel ./test.trig scott net1
```

The expected output of the `java` command might appear as follows:

```
solution
[p=http://xmlns.com/foaf/0.1/mbox;s=http://example.org/alice;g=http://
example.org/aliceGraph;o=mailto:alice@work.example.org]
solution
[p=http://xmlns.com/foaf/0.1/name;s=http://example.org/alice;g=http://
example.org/aliceGraph;o="Alice"]
solution
[p=http://xmlns.com/foaf/0.1/mbox;s=http://example.org/bob;g=http://
example.org/bobGraph;o=mailto:bob@oldcorp.example.org]
total # of solution 3
```

8.14.9 Example 9: Get COUNT of Matches

Example 8-13 shows the `CountQuery.java` file, which demonstrates how to perform a query that returns the total number (COUNT) of matches.

Example 8-13 Get COUNT of Matches

```
import java.io.PrintStream;
import java.sql.SQLException;
import oracle.rdf4j.adapter.OraclePool;
import oracle.rdf4j.adapter.OracleRepository;
```

```
import oracle.rdf4j.adapter.OracleSailConnection;
import oracle.rdf4j.adapter.OracleSailRepositoryConnection;
import oracle.rdf4j.adapter.OracleSailStore;
import oracle.rdf4j.adapter.exception.ConnectionSetupException;
import oracle.rdf4j.adapter.utils.OracleUtils;
import org.eclipse.rdf4j.model.IRI;
import org.eclipse.rdf4j.model.Literal;
import org.eclipse.rdf4j.model.ValueFactory;
import org.eclipse.rdf4j.model.vocabulary.RDF;
import org.eclipse.rdf4j.query.BindingSet;
import org.eclipse.rdf4j.query.QueryLanguage;
import org.eclipse.rdf4j.query.TupleQuery;
import org.eclipse.rdf4j.query.TupleQueryResult;
import org.eclipse.rdf4j.repository.Repository;
import org.eclipse.rdf4j.repository.RepositoryConnection;

public class CountQuery {
  public static void main(String[] args) throws
    ConnectionSetupException, SQLException
  {
    PrintStream psOut = System.out;
    String jdbcUrl = args[0];
    String user = args[1];
    String password = args[2];
    String model = args[3];
    String networkOwner = (args.length > 5) ? args[4] : null;
    String networkName = (args.length > 5) ? args[5] : null;


    OraclePool op = null;
    OracleSailStore store = null;
    Repository sr = null;
    RepositoryConnection conn = null;

    try {
      op = new OraclePool(jdbcUrl, user, password);
      store = new OracleSailStore(op, model, networkOwner, networkName);
      sr = new OracleRepository(store);
      conn = sr.getConnection();

      ValueFactory f = conn.getValueFactory();

      // create some resources and literals to make statements out of
      IRI alice = f.createIRI("http://example.org/people/alice");
      IRI name = f.createIRI("http://example.org/ontology/name");
      IRI person = f.createIRI("http://example.org/ontology/Person");
      Literal alicesName = f.createLiteral("Alice");

      conn.begin();
      // clear model to start fresh
      conn.clear();
      conn.add(alice, RDF.TYPE, person);
      conn.add(alice, name, alicesName);
      conn.commit();

      OracleSailConnection osc =
```

```
        (OracleSailConnection)((OracleSailRepositoryConnection)
conn).getSailConnection();
      osc.analyze();
      osc.analyzeApplicationTable();

      // Run a query and only return the number of matches (the count ! )
      String queryString = " SELECT (COUNT(*) AS ?totalCount) WHERE {?s ?p ?
y} ";

      TupleQuery tupleQuery = conn.prepareTupleQuery(
      QueryLanguage.SPARQL, queryString);

      try (TupleQueryResult result = tupleQuery.evaluate()) {
        if (result.hasNext()) {
          BindingSet bindingSet = result.next();
          String totalCount = bindingSet.getValue("totalCount").stringValue();
          psOut.println("number of matches: " + totalCount);
        }
      }
    }
    finally {
      if (conn != null && conn.isOpen()) {
        conn.clear();
        conn.close();
      }
      OracleUtils.dropSemanticModelAndTables(op.getOracleDB(), model, null,
null, networkOwner, networkName);
      sr.shutDown();
      store.shutDown();
      op.close();
    }
  }
}
```

To compile this example, execute the following command:

```
javac -classpath $CP CountQuery.java
```

To run this example for an existing schema-private network whose owner is SCOTT and name is NET1, execute the following command:

```
java -classpath $CP CountQuery jdbc:oracle:thin:@localhost:1521:ORCL scott
<password> TestModel scott net1
```

The expected output of the `java` command might appear as follows:

```
number of matches: 2
```

8.14.10 Example 10: Specify Bind Variable for Constant in Query Pattern

Example 8-13 shows the `BindVar.java` file, which demonstrates how to perform a query that specifies a bind variable for a constant in the SPARQL query pattern.

Example 8-14 Specify Bind Variable for Constant in Query Pattern

```
import java.io.PrintStream;
import java.sql.SQLException;
import oracle.rdf4j.adapter.OraclePool;
import oracle.rdf4j.adapter.OracleRepository;
import oracle.rdf4j.adapter.OracleSailStore;
import oracle.rdf4j.adapter.exception.ConnectionSetupException;
import oracle.rdf4j.adapter.utils.OracleUtils;
import org.eclipse.rdf4j.model.IRI;
import org.eclipse.rdf4j.model.Literal;
import org.eclipse.rdf4j.model.ValueFactory;
import org.eclipse.rdf4j.model.vocabulary.RDF;
import org.eclipse.rdf4j.query.BindingSet;
import org.eclipse.rdf4j.query.QueryLanguage;
import org.eclipse.rdf4j.query.TupleQuery;
import org.eclipse.rdf4j.query.TupleQueryResult;
import org.eclipse.rdf4j.repository.Repository;
import org.eclipse.rdf4j.repository.RepositoryConnection;

public class BindVar {
  public static void main(String[] args) throws ConnectionSetupException,
SQLException {
    PrintStream psOut = System.out;
    String jdbcUrl = args[0];
    String user = args[1];
    String password = args[2];
    String model = args[3];
    String networkOwner = (args.length > 5) ? args[4] : null;
    String networkName = (args.length > 5) ? args[5] : null;


    OraclePool op = null;
    OracleSailStore store = null;
    Repository sr = null;
    RepositoryConnection conn = null;

    try {
      op = new OraclePool(jdbcUrl, user, password);
      store = new OracleSailStore(op, model, networkOwner, networkName);
      sr = new OracleRepository(store);
      conn = sr.getConnection();
      ValueFactory f = conn.getValueFactory();

      conn.begin();
      conn.clear();

      // create some resources and literals to make statements out of

      // Alice
      IRI alice = f.createIRI("http://example.org/people/alice");
      IRI name = f.createIRI("http://example.org/ontology/name");
      IRI person = f.createIRI("http://example.org/ontology/Person");
      Literal alicesName = f.createLiteral("Alice");
      conn.add(alice, RDF.TYPE, person);
```

```
      conn.add(alice, name, alicesName);

      //Bob
      IRI bob = f.createIRI("http://example.org/people/bob");
      Literal bobsName = f.createLiteral("Bob");
      conn.add(bob, RDF.TYPE, person);
      conn.add(bob, name, bobsName);

      conn.commit();

      String queryString =
        " PREFIX ex: <http://example.org/ontology/> " +
        " Select ?name \n" + " WHERE \n" + " { SELECT * WHERE { ?person
ex:name ?name} }\n" +
        " ORDER BY ?name";

      TupleQuery tupleQuery = conn.prepareTupleQuery(
      QueryLanguage.SPARQL, queryString);

      // set binding for ?person = Alice
      tupleQuery.setBinding("person", alice);
      try (TupleQueryResult result = tupleQuery.evaluate()) {
        if (result.hasNext()) {
          BindingSet bindingSet = result.next();
          psOut.println("solution " + bindingSet.toString());
        }
      }

      // re-run with ?person = Bob
      tupleQuery.setBinding("person", bob);
      try (TupleQueryResult result = tupleQuery.evaluate()) {
        if (result.hasNext()) {
          BindingSet bindingSet = result.next();
          psOut.println("solution " + bindingSet.toString());
        }
      }
    }
    finally {
      if (conn != null && conn.isOpen()) {
        conn.clear();
        conn.close();
      }
      OracleUtils.dropSemanticModelAndTables(op.getOracleDB(), model, null,
null, networkOwner, networkName);
      sr.shutDown();
      store.shutDown();
      op.close();
    }
  }
}
```

To compile this example, execute the following command:

```
javac -classpath $CP BindVar.java
```

To run this example for an existing schema-private network whose owner is SCOTT and name is NET1, execute the following command:

```
java -classpath $CP BindVar jdbc:oracle:thin:@localhost:1521:ORCL scott
<password> TestModel scott net1
```

The expected output of the `java` command might appear as follows:

```
solution [name="Alice";person=http://example.org/people/alice]
solution [name="Bob";person=http://example.org/people/bob]
```

8.14.11 Example 11: SPARQL Update

Example 8-15 shows the `SparqlUpdate.java` file, which demonstrates how to perform SPARQL Update statements.

Example 8-15 SPARQL Update

```
import java.io.PrintStream;
import java.sql.SQLException;
import oracle.rdf4j.adapter.OraclePool;
import oracle.rdf4j.adapter.OracleRepository;
import oracle.rdf4j.adapter.OracleSailStore;
import oracle.rdf4j.adapter.exception.ConnectionSetupException;
import oracle.rdf4j.adapter.utils.OracleUtils;
import org.eclipse.rdf4j.query.BindingSet;
import org.eclipse.rdf4j.query.QueryLanguage;
import org.eclipse.rdf4j.query.TupleQuery;
import org.eclipse.rdf4j.query.TupleQueryResult;
import org.eclipse.rdf4j.query.Update;
import org.eclipse.rdf4j.repository.Repository;
import org.eclipse.rdf4j.repository.RepositoryConnection;

public class SparqlUpdate {
  private static final String DATA_1 =
    "[p=http://example.org/ontology/name;g=urn:g1;x=http://example.org/people/
Sue;y=\"Sue\"]" +
    "[p=http://www.w3.org/1999/02/22-rdf-syntax-ns#type;g=urn:g1;x=http://
example.org/people/Sue;y=http://example.org/ontology/Person]";

  private static final String DATA_2 =
    "[p=http://example.org/ontology/name;g=urn:g1;x=http://example.org/people/
Sue;y=\"Susan\"]" +
    "[p=http://www.w3.org/1999/02/22-rdf-syntax-ns#type;g=urn:g1;x=http://
example.org/people/Sue;y=http://example.org/ontology/Person]";

  private static final String DATA_3 =
    "[p=http://example.org/ontology/name;g=urn:g1;x=http://example.org/people/
Sue;y=\"Susan\"]" +
    "[p=http://www.w3.org/1999/02/22-rdf-syntax-ns#type;g=urn:g1;x=http://
example.org/people/Sue;y=http://example.org/ontology/Person]" +
    "[p=http://example.org/ontology/name;g=urn:g2;x=http://example.org/people/
Sue;y=\"Susan\"]" +
    "[p=http://www.w3.org/1999/02/22-rdf-syntax-ns#type;g=urn:g2;x=http://
```

```
example.org/people/Sue;y=http://example.org/ontology/Person]";

  private static final String DATA_4 =
    "[p=http://www.w3.org/1999/02/22-rdf-syntax-ns#type;g=urn:g1;x=http://
example.org/people/Sue;y=http://example.org/ontology/Person]" +
    "[p=http://example.org/ontology/name;g=urn:g2;x=http://example.org/people/
Sue;y=\"Susan\"]" +
    "[p=http://www.w3.org/1999/02/22-rdf-syntax-ns#type;g=urn:g2;x=http://
example.org/people/Sue;y=http://example.org/ontology/Person]";

  private static final String DATA_5 =
    "[p=http://example.org/ontology/name;g=urn:g1;x=http://example.org/people/
Sue;y=\"Susan\"]" +
    "[p=http://www.w3.org/1999/02/22-rdf-syntax-ns#type;g=urn:g1;x=http://
example.org/people/Sue;y=http://example.org/ontology/Person]" +
    "[p=http://example.org/ontology/name;g=urn:g2;x=http://example.org/people/
Sue;y=\"Susan\"]" +
    "[p=http://www.w3.org/1999/02/22-rdf-syntax-ns#type;g=urn:g2;x=http://
example.org/people/Sue;y=http://example.org/ontology/Person]";

  private static String getRepositoryData(RepositoryConnection conn,
PrintStream out)
  {
    String dataStr = "";
    String queryString = "SELECT * WHERE { GRAPH ?g { ?x ?p ?y } } ORDER BY ?
g ?x ?p ?y";
    TupleQuery tupleQuery = conn.prepareTupleQuery(QueryLanguage.SPARQL,
queryString);
    try (TupleQueryResult result = tupleQuery.evaluate()) {
      while (result.hasNext()) {
        BindingSet bindingSet = result.next();
        out.println(bindingSet.toString());
        dataStr += bindingSet.toString();
      }
    }
    return dataStr;
  }
  public static void main(String[] args) throws
    ConnectionSetupException, SQLException
  {
    PrintStream out = new PrintStream(System.out);
    String jdbcUrl = args[0];
    String user = args[1];
    String password = args[2];
    String model = args[3];
    String networkOwner = (args.length > 5) ? args[4] : null;
    String networkName = (args.length > 5) ? args[5] : null;


    OraclePool op = null;
    OracleSailStore store = null;
    Repository sr = null;
    RepositoryConnection conn = null;
    try {
      op = new OraclePool(jdbcUrl, user, password);
      store = new OracleSailStore(op, model, networkOwner, networkName);
```

```
        sr = new OracleRepository(store);
        conn = sr.getConnection();

        conn.clear(); // to start from scratch

        // Insert some initial data
        String updString = "PREFIX people: <http://example.org/people/>\n" +
                           "PREFIX    ont: <http://example.org/ontology/>\n" +
                           "INSERT DATA { GRAPH <urn:g1> { \n" +
                           "               people:Sue a ont:Person; \n" +
                           "                 ont:name \"Sue\" . } }";
        Update upd = conn.prepareUpdate(QueryLanguage.SPARQL, updString);
        upd.execute();
        conn.commit();
        String repositoryData = getRepositoryData(conn, out);
        if (! (DATA_1.equals(repositoryData)) ) out.println("DATA_1 mismatch");
        // Change Sue's name to Susan
        updString = "PREFIX people: <http://example.org/people/>\n" +
                    "PREFIX    ont: <http://example.org/ontology/>\n" +
                    "DELETE { GRAPH ?g { ?s ont:name ?n } }\n" +
                    "INSERT { GRAPH ?g { ?s ont:name \"Susan\" } }\n" +
                    "WHERE  { GRAPH ?g { ?s ont:name ?n FILTER (?n =
\"Sue\") }}";
        upd = conn.prepareUpdate(QueryLanguage.SPARQL, updString);
        upd.execute();
        conn.commit();
        repositoryData = getRepositoryData(conn, out);
        if (! (DATA_2.equals(repositoryData)) ) out.println("DATA_2 mismatch");

        // Copy to contents of g1 to a new graph g2
        updString = "PREFIX people: <http://example.org/people/>\n" +
                    "PREFIX ont: <http://example.org/ontology/>\n" +
                    "COPY <urn:g1> TO <urn:g2>";
        upd = conn.prepareUpdate(QueryLanguage.SPARQL, updString);
        upd.execute();
        conn.commit();

        repositoryData = getRepositoryData(conn, out);
        if (! (DATA_3.equals(repositoryData)) ) out.println("DATA_3 mismatch");

        // Delete ont:name triple from graph g1
        updString = "PREFIX people: <http://example.org/people/>\n" +
                    "PREFIX  ont: <http://example.org/ontology/>\n" +
                    "DELETE DATA { GRAPH <urn:g1> { people:Sue ont:name
\"Susan\" } }";
        upd = conn.prepareUpdate(QueryLanguage.SPARQL, updString);
        upd.execute();
        conn.commit();
        repositoryData = getRepositoryData(conn, out);
        if (! (DATA_4.equals(repositoryData)) ) out.println("DATA_4 mismatch");

        // Add contents of g2 to g1
        updString = "PREFIX people: <http://example.org/people/>\n" +
                    "PREFIX    ont: <http://example.org/ontology/>\n" +
                    "ADD <urn:g2> TO <urn:g1>";
        upd = conn.prepareUpdate(QueryLanguage.SPARQL, updString);
```

```
      upd.execute();
      conn.commit();
      repositoryData = getRepositoryData(conn, out);
      if (! (DATA_5.equals(repositoryData)) ) out.println("DATA_5 mismatch");
    }
    finally {
      if (conn != null && conn.isOpen()) {
        conn.clear();
        conn.close();
      }
      OracleUtils.dropSemanticModelAndTables(op.getOracleDB(), model, null,
null, networkOwner, networkName);
      sr.shutDown();
      store.shutDown();
      op.close();
    }
  }
}
```

To compile this example, execute the following command:

```
javac -classpath $CP SparqlUpdate.java
```

To run this example for an existing schema-private network whose owner is SCOTT and name is NET1, execute the following command:

```
java -classpath $CP SparqlUpdate jdbc:oracle:thin:@localhost:1521:ORCL scott
<password> TestModel scott net1
```

The expected output of the `java` command might appear as follows:

```
[p=http://example.org/ontology/name;g=urn:g1;x=http://example.org/people/
Sue;y="Sue"]
[p=http://www.w3.org/1999/02/22-rdf-syntax-ns#type;g=urn:g1;x=http://
example.org/people/Sue;y=http://example.org/ontology/Person]
[p=http://example.org/ontology/name;g=urn:g1;x=http://example.org/people/
Sue;y="Susan"]
[p=http://www.w3.org/1999/02/22-rdf-syntax-ns#type;g=urn:g1;x=http://
example.org/people/Sue;y=http://example.org/ontology/Person]
[p=http://example.org/ontology/name;g=urn:g1;x=http://example.org/people/
Sue;y="Susan"]
[p=http://www.w3.org/1999/02/22-rdf-syntax-ns#type;g=urn:g1;x=http://
example.org/people/Sue;y=http://example.org/ontology/Person]
[p=http://example.org/ontology/name;g=urn:g2;x=http://example.org/people/
Sue;y="Susan"]
[p=http://www.w3.org/1999/02/22-rdf-syntax-ns#type;g=urn:g2;x=http://
example.org/people/Sue;y=http://example.org/ontology/Person]
[p=http://www.w3.org/1999/02/22-rdf-syntax-ns#type;g=urn:g1;x=http://
example.org/people/Sue;y=http://example.org/ontology/Person]
[p=http://example.org/ontology/name;g=urn:g2;x=http://example.org/people/
Sue;y="Susan"]
[p=http://www.w3.org/1999/02/22-rdf-syntax-ns#type;g=urn:g2;x=http://
example.org/people/Sue;y=http://example.org/ontology/Person]
[p=http://example.org/ontology/name;g=urn:g1;x=http://example.org/people/
```

```
Sue;y="Susan"]
[p=http://www.w3.org/1999/02/22-rdf-syntax-ns#type;g=urn:g1;x=http://
example.org/people/Sue;y=http://example.org/ontology/Person]
[p=http://example.org/ontology/name;g=urn:g2;x=http://example.org/people/
Sue;y="Susan"]
[p=http://www.w3.org/1999/02/22-rdf-syntax-ns#type;g=urn:g2;x=http://
example.org/people/Sue;y=http://example.org/ontology/Person]
```

8.14.12 Example 12: Oracle Hint

Example 8-16 shows the `OracleHint.java` file, which demonstrates how to use Oracle hint in a SPARQL query or a SPARQL update.

Example 8-16 Oracle Hint

```
import java.sql.SQLException;
import oracle.rdf4j.adapter.OracleDB;
import oracle.rdf4j.adapter.OraclePool;
import oracle.rdf4j.adapter.OracleRepository;
import oracle.rdf4j.adapter.OracleSailStore;
import oracle.rdf4j.adapter.exception.ConnectionSetupException;
import oracle.rdf4j.adapter.utils.OracleUtils;
import org.eclipse.rdf4j.query.BindingSet;
import org.eclipse.rdf4j.query.QueryLanguage;
import org.eclipse.rdf4j.query.TupleQuery;
import org.eclipse.rdf4j.query.TupleQueryResult;
import org.eclipse.rdf4j.query.Update;
import org.eclipse.rdf4j.repository.Repository;
import org.eclipse.rdf4j.repository.RepositoryConnection;

public class OracleHint {
  public static void main(String[] args) throws ConnectionSetupException,
SQLException {
    String jdbcUrl = args[0];
    String user = args[1];
    String password = args[2];
    String model = args[3];
    String networkOwner = (args.length > 5) ? args[4] : null;
    String networkName = (args.length > 5) ? args[5] : null;


    OraclePool op = null;
    OracleSailStore store = null;
    Repository sr = null;
    RepositoryConnection conn = null;

    try {
      op = new OraclePool(jdbcUrl, user, password);
      store = new OracleSailStore(op, model, networkOwner, networkName);
      sr = new OracleRepository(store);
      conn = sr.getConnection();

      conn.clear(); // to start from scratch

      // Insert some initial data
```

```
      String updString =
        "PREFIX ex: <http://example.org/>\n" +
        "INSERT DATA {  " +
        "  ex:a ex:p1 ex:b . " +
        "  ex:b ex:p1 ex:c . " +
        "  ex:c ex:p1 ex:d . " +
        "  ex:d ex:p1 ex:e . " +
        "  ex:e ex:p1 ex:f . " +
        "  ex:f ex:p1 ex:g . " +
        "  ex:g ex:p1 ex:h . " +
        "  ex:h ex:p1 ex:i . " +
        "  ex:i ex:p1 ex:j . " +
        "  ex:j ex:p1 ex:k . " +
        "}";
      Update upd = conn.prepareUpdate(QueryLanguage.SPARQL, updString);
      upd.execute();
      conn.commit();

      // default behavior for property path is 10 hop max, so we get 11
results
      String sparql =
        "PREFIX ex: <http://example.org/>\n" +
        "SELECT (COUNT(*) AS ?cnt)\n" +
        "WHERE { ex:a ex:p1* ?y }";

      TupleQuery tupleQuery = conn.prepareTupleQuery(QueryLanguage.SPARQL,
sparql);

      try (TupleQueryResult result = tupleQuery.evaluate()) {
        while (result.hasNext()) {
          BindingSet bindingSet = result.next();
          if (11 !=
Integer.parseInt(bindingSet.getValue("cnt").stringValue()))
System.out.println("cnt mismatch: expecting 11");
        }
      }

      // ORACLE_SEM_FS_NS prefix hint to use parallel(2) and
dynamic_sampling(6)
      // ORACLE_SEM_SM_NS prefix hint to use a 5 hop max and to use CONNECT
BY instead of simple join
      sparql =
        "PREFIX ORACLE_SEM_FS_NS: <http://oracle.com/semtech#dop=2,ods=6>\n" +
        "PREFIX ORACLE_SEM_SM_NS: <http://oracle.com/
semtech#all_max_pp_depth(5),all_disable_pp_sj>\n" +
        "PREFIX ex: <http://example.org/>\n" +
        "SELECT (COUNT(*) AS ?cnt)\n" +
        "WHERE { ex:a ex:p1* ?y }";

      tupleQuery = conn.prepareTupleQuery(QueryLanguage.SPARQL, sparql,
"http://example.org/");

      try (TupleQueryResult result = tupleQuery.evaluate()) {
        while (result.hasNext()) {
          BindingSet bindingSet = result.next();
          if (6 !=
```

```
Integer.parseInt(bindingSet.getValue("cnt").stringValue()))
System.out.println("cnt mismatch: expecting 6");
        }
      }

      // query options for SPARQL Update
      sparql =
        "PREFIX ORACLE_SEM_UM_NS: <http://oracle.com/semtech#parallel(2)>\n" +
        "PREFIX ORACLE_SEM_SM_NS: <http://oracle.com/
semtech#all_max_pp_depth(5),all_disable_pp_sj>\n" +
        "PREFIX ex: <http://example.org/>\n" +
        "INSERT { GRAPH ex:g1 { ex:a ex:reachable ?y } }\n" +
        "WHERE { ex:a ex:p1* ?y }";

      Update u = conn.prepareUpdate(sparql);
      u.execute();

      // graph ex:g1 should have 6 results because of all_max_pp_depth(5)
      sparql =
        "PREFIX ex: <http://example.org/>\n" +
        "SELECT (COUNT(*) AS ?cnt)\n" +
        "WHERE { GRAPH ex:g1 { ?s ?p ?o } }";

      tupleQuery = conn.prepareTupleQuery(QueryLanguage.SPARQL, sparql,
"http://example.org/");

      try (TupleQueryResult result = tupleQuery.evaluate()) {
        while (result.hasNext()) {
          BindingSet bindingSet = result.next();
          if (6 !=
Integer.parseInt(bindingSet.getValue("cnt").stringValue()))
System.out.println("cnt mismatch: expecting 6");
        }
      }
    }
    finally {
      if (conn != null && conn.isOpen()) {
        conn.clear();
        conn.close();
      }
      OracleUtils.dropSemanticModelAndTables(op.getOracleDB(), model, null,
null, networkOwner, networkName);
      sr.shutDown();
      store.shutDown();
      op.close();
    }
  }
}
```

To compile this example, execute the following command:

```
javac -classpath $CP OracleHint.java
```

To run this example for an existing schema-private network whose owner is SCOTT and name is NET1, execute the following command:

```
java -classpath $CP OracleHint jdbc:oracle:thin:@localhost:1521:ORCL scott
<password> TestModel scott net1
```

8.14.13 Example 13: Using JDBC Bind Values

Example 8-17 shows the `JDBCBindVar.java` file, which demonstrates how to use JDBC bind values.

Example 8-17   Using JDBC Bind Values

```
import java.io.PrintStream;
import java.sql.SQLException;
import oracle.rdf4j.adapter.OracleDB;
import oracle.rdf4j.adapter.OraclePool;
import oracle.rdf4j.adapter.OracleRepository;
import oracle.rdf4j.adapter.OracleSailStore;
import oracle.rdf4j.adapter.exception.ConnectionSetupException;
import oracle.rdf4j.adapter.utils.OracleUtils;
import org.eclipse.rdf4j.model.IRI;
import org.eclipse.rdf4j.model.Literal;
import org.eclipse.rdf4j.model.ValueFactory;
import org.eclipse.rdf4j.model.vocabulary.RDF;
import org.eclipse.rdf4j.query.BindingSet;
import org.eclipse.rdf4j.query.QueryLanguage;
import org.eclipse.rdf4j.query.TupleQuery;
import org.eclipse.rdf4j.query.TupleQueryResult;
import org.eclipse.rdf4j.repository.Repository;
import org.eclipse.rdf4j.repository.RepositoryConnection;

public class JDBCBindVar {

  public static void main(String[] args) throws ConnectionSetupException,
SQLException {
    PrintStream psOut = System.out;

    String jdbcUrl = args[0];
    String user = args[1];
    String password = args[2];
    String model = args[3];
    String networkOwner = (args.length > 5) ? args[4] : null;
    String networkName = (args.length > 5) ? args[5] : null;
    OraclePool op = null;
    OracleSailStore store = null;
    Repository sr = null;
    RepositoryConnection conn = null;

    try {
      op = new OraclePool(jdbcUrl, user, password);
      store = (networkName == null) ? new OracleSailStore(op, model) : new
OracleSailStore(op, model, networkOwner, networkName);
      sr = new OracleRepository(store);
      conn = sr.getConnection();
```

```
      ValueFactory f = conn.getValueFactory();

      conn.begin();
      conn.clear();

      // create some resources and literals to make statements out of
      // Alice
      IRI alice = f.createIRI("http://example.org/people/alice");
      IRI name = f.createIRI("http://example.org/ontology/name");
      IRI person = f.createIRI("http://example.org/ontology/Person");
      Literal alicesName = f.createLiteral("Alice");
      conn.add(alice, RDF.TYPE, person);
      conn.add(alice, name, alicesName);

      //Bob
      IRI bob = f.createIRI("http://example.org/people/bob");
      Literal bobsName = f.createLiteral("Bob");
      conn.add(bob, RDF.TYPE, person);
      conn.add(bob, name, bobsName);

      conn.commit();

      // Query using USE_BIND_VAR=JDBC option for JDBC bind values
      // Simple BIND clause for ?person marks ?person as a bind variable
      String queryString =
        " PREFIX ORACLE_SEM_SM_NS: <http://oracle.com/
semtech#USE_BIND_VAR=JDBC>\n" +
        " PREFIX ex: <http://example.org/ontology/>\n" +
        " Select ?name \n" +
        " WHERE \n" +
        " { SELECT * WHERE { \n" +
        "     BIND (\"\" AS ?person) \n" +
        "     ?person ex:name ?name } \n" +
        " }\n" +
        " ORDER BY ?name";
      TupleQuery tupleQuery = conn.prepareTupleQuery(
          QueryLanguage.SPARQL, queryString);

      // set binding for ?person = Alice
      tupleQuery.setBinding("person", alice);
      try (TupleQueryResult result = tupleQuery.evaluate()) {
        if (result.hasNext()) {
          BindingSet bindingSet = result.next();
          psOut.println("solution " + bindingSet.toString());
        }
      }

      // re-run with ?person = Bob
      tupleQuery.setBinding("person", bob);
      try (TupleQueryResult result = tupleQuery.evaluate()) {
        if (result.hasNext()) {
          BindingSet bindingSet = result.next();
          psOut.println("solution " + bindingSet.toString());
        }
      }
```

```
    }
    finally {
      if (conn != null && conn.isOpen()) {
        conn.clear();
        conn.close();
      }
      if (op != null) {
        OracleDB oracleDB = op.getOracleDB();
        if (networkName == null)
          OracleUtils.dropSemanticModelAndTables(oracleDB, model);
        else
          OracleUtils.dropSemanticModelAndTables(oracleDB, model, null, null,
networkOwner, networkName);
        op.returnOracleDBtoPool(oracleDB);
      }
      sr.shutDown();
      store.shutDown();
      op.close();
      }
  }
}
```

To compile this example, execute the following command:

```
javac -classpath $CP JDBCBindVar.java
```

To run this example for an existing schema-private network whose owner is SCOTT and name is NET1, execute the following command:

```
java -classpath $CP JDBCBindVar jdbc:oracle:thin:@localhost:1521:ORCL scott
<password-for-scott> TestModel scott net1
```

The expected output of the Java command might appear as follows:

```
solution [name="Alice";person=http://example.org/people/alice]
solution [name="Bob";person=http://example.org/people/bob]
```

8.14.14 Example 14: Simple Inference

Example 8-18 shows the `SimpleInference.java` file, which shows inference for a single RDF graph using the OWL2RL rule base.

Example 8-18 Simple Inference

```
import java.io.IOException;
import java.io.PrintStream;
import java.sql.SQLException;
import oracle.rdf4j.adapter.OraclePool;
import oracle.rdf4j.adapter.OracleRepository;
import oracle.rdf4j.adapter.OracleSailStore;
import oracle.rdf4j.adapter.OracleSailConnection;
import oracle.rdf4j.adapter.exception.ConnectionSetupException;
import oracle.rdf4j.adapter.utils.OracleUtils;
```

```
import org.eclipse.rdf4j.model.IRI;
import org.eclipse.rdf4j.model.Literal;
import org.eclipse.rdf4j.model.ValueFactory;
import org.eclipse.rdf4j.model.vocabulary.RDF;
import org.eclipse.rdf4j.model.vocabulary.RDFS;
import org.eclipse.rdf4j.query.BindingSet;
import org.eclipse.rdf4j.query.QueryLanguage;
import org.eclipse.rdf4j.query.TupleQuery;
import org.eclipse.rdf4j.query.TupleQueryResult;
import org.eclipse.rdf4j.repository.Repository;
import org.eclipse.rdf4j.repository.RepositoryConnection;
import oracle.rdf4j.adapter.Attachment;
import oracle.rdf4j.adapter.OracleSailConnection;
import oracle.rdf4j.adapter.OracleSailRepositoryConnection;

public class SimpleInference {
  public static void main(String[] args) throws ConnectionSetupException,
SQLException, IOException {
    PrintStream psOut = System.out;
    String jdbcUrl = args[0];
    String user = args[1];
    String password = args[2];
    String model = args[3];
    String networkOwner = (args.length > 5) ? args[4] : null;
    String networkName = (args.length > 5) ? args[5] : null;

    OraclePool op = null;
    OracleSailStore store = null;
    Repository sr = null;
    RepositoryConnection conn = null;

    try {
      op = new OraclePool(jdbcUrl, user, password);

      // create a single-model, single-rulebase OracleSailStore object
      Attachment attachment =
Attachment.createInstance(Attachment.NO_ADDITIONAL_MODELS, new String[]
{"OWL2RL"});
      store = new OracleSailStore(op, model, attachment, networkOwner,
networkName);
      sr = new OracleRepository(store);

      ValueFactory f = sr.getValueFactory();
      conn = sr.getConnection();

      // create some resources and literals to make statements out of
      IRI alice = f.createIRI("http://example.org/people/alice");
      IRI bob = f.createIRI("http://example.org/people/bob");
      IRI friendOf = f.createIRI("http://example.org/ontology/friendOf");
      IRI Person = f.createIRI("http://example.org/ontology/Person");
      IRI Engineer = f.createIRI("http://example.org/ontology/Engineer");
      IRI Doctor = f.createIRI("http://example.org/ontology/Doctor");

      conn.clear(); // to start from scratch

      // add some statements to the RDF graph
```

```
      conn.add(alice, RDF.TYPE, Engineer);
      conn.add(bob, RDF.TYPE, Doctor);
      conn.add(alice, friendOf, bob);
      conn.commit();

      OracleSailConnection osc = (OracleSailConnection)
((OracleSailRepositoryConnection)conn).getSailConnection();

      // perform inference (this will not generate any inferred triples)
      osc.performInference();

      // prepare a query to run against the repository
      String queryString =
        "PREFIX ex: <http://example.org/ontology/>\n" +
        "SELECT * WHERE {?x ex:friendOf ?y . ?x a ex:Person . ?y a ex:Person}
\n" ;
      TupleQuery tupleQuery = conn.prepareTupleQuery(QueryLanguage.SPARQL,
queryString);

      // run the query: no results will be returned because nobody is a Person
      try (TupleQueryResult result = tupleQuery.evaluate()) {
        int resultCount = 0;
        while (result.hasNext()) {
          resultCount++;
          BindingSet bindingSet = result.next();
          psOut.println("value of x: " + bindingSet.getValue("x"));
          psOut.println("value of y: " + bindingSet.getValue("y"));
        }
        psOut.println("number of results: " + resultCount);
      }

      // add class hierarchy
      conn.add(Doctor, RDFS.SUBCLASSOF, Person);
      conn.add(Engineer, RDFS.SUBCLASSOF, Person);
      conn.commit();

      // perform inference again
      osc.performInference();

      // run the same query again: returns some results because alice and bob
now belong to superclass Person
      try (TupleQueryResult result = tupleQuery.evaluate()) {
        while (result.hasNext()) {
          BindingSet bindingSet = result.next();
          psOut.println("value of x: " + bindingSet.getValue("x"));
          psOut.println("value of y: " + bindingSet.getValue("y"));
        }
      }
    }
    finally {
      if (conn != null && conn.isOpen()) {
        conn.clear();
        conn.close();
      }
      OracleUtils.dropSemanticModelAndTables(op.getOracleDB(), model, null,
null, networkOwner, networkName);
```

```
      sr.shutDown();
      store.shutDown();
      op.close();
    }
  }
}
```

To compile this example, execute the following command:

```
javac -classpath $CP SimpleInference.java
```

To run this example for an existing schema-private network whose owner is SCOTT and name is NET1, execute the following command:

```
java -classpath $CP SimpleInference jdbc:oracle:thin:@localhost:1521:ORCL
scott <password-for-scott> TestModel scott net1
```

The expected output of the Java command might appear as follows:

```
number of results: 0
value of x: http://example.org/people/alice
value of y: http://example.org/people/bob
```

8.14.15 Example 15: Simple Graph Collection

Example 8-19 shows the `SimpleVirtualModel.java` file, which shows the creation and use of an RDF graph collection consisting of two RDF graphs.

Example 8-19 Simple Graph Collection

```
import java.io.IOException;
import java.io.PrintStream;
import java.sql.SQLException;
import oracle.rdf4j.adapter.OraclePool;
import oracle.rdf4j.adapter.OracleRepository;
import oracle.rdf4j.adapter.OracleSailStore;
import oracle.rdf4j.adapter.exception.ConnectionSetupException;
import oracle.rdf4j.adapter.utils.OracleUtils;
import org.eclipse.rdf4j.model.IRI;
import org.eclipse.rdf4j.model.ValueFactory;
import org.eclipse.rdf4j.model.vocabulary.RDF;
import org.eclipse.rdf4j.model.vocabulary.RDFS;
import org.eclipse.rdf4j.query.BindingSet;
import org.eclipse.rdf4j.query.QueryLanguage;
import org.eclipse.rdf4j.query.TupleQuery;
import org.eclipse.rdf4j.query.TupleQueryResult;
import org.eclipse.rdf4j.repository.Repository;
import org.eclipse.rdf4j.repository.RepositoryConnection;
import oracle.rdf4j.adapter.Attachment;

public class SimpleVirtualModel {
  public static void main(String[] args) throws ConnectionSetupException,
SQLException, IOException {
```

```
    PrintStream psOut = System.out;
    String jdbcUrl = args[0];
    String user = args[1];
    String password = args[2];
    String model = args[3];
    String model2 = args[4];
    String virtualModelName = args[5];
    String networkOwner = (args.length > 7) ? args[6] : null;
    String networkName = (args.length > 7) ? args[7] : null;

    OraclePool op = null;

    OracleSailStore store = null;
    Repository sr = null;
    RepositoryConnection conn = null;

    OracleSailStore store2 = null;
    Repository sr2 = null;
    RepositoryConnection conn2 = null;

    OracleSailStore vmStore = null;
    Repository vmSr = null;
    RepositoryConnection vmConn = null;

    try {
      op = new OraclePool(jdbcUrl, user, password);

      // create two RDF graphs and then an RDF graph collection that uses
those two graphs

      // create the first RDF grapj
      store = new OracleSailStore(op, model, networkOwner, networkName);
      sr = new OracleRepository(store);
      ValueFactory f = sr.getValueFactory();
      conn = sr.getConnection();

      // create the second RDF graph (this one will be used as an additional
graph in the attachment object)
      store2 = new OracleSailStore(op, model2, networkOwner, networkName);
      sr2 = new OracleRepository(store2);
      conn2 = sr2.getConnection();

      // create a two-graph RDF graph collection OracleSailStore object
      Attachment attachment = Attachment.createInstance(model2);
      vmStore = new OracleSailStore(op, model, /*ignored*/true, /
*useVirtualModel*/true, virtualModelName, attachment, networkOwner,
networkName);
      vmSr = new OracleRepository(vmStore);
      vmConn = vmSr.getConnection();

      // create some resources and literals to make statements out of
      IRI alice = f.createIRI("http://example.org/people/alice");
      IRI bob = f.createIRI("http://example.org/people/bob");
      IRI friendOf = f.createIRI("http://example.org/ontology/friendOf");
      IRI Person = f.createIRI("http://example.org/ontology/Person");
      IRI Engineer = f.createIRI("http://example.org/ontology/Engineer");
```

```
      IRI Doctor = f.createIRI("http://example.org/ontology/Doctor");

      // clear any data (in case any of the two non-virtual models were
already present)
      conn.clear();
      conn2.clear();

      // add some statements to the first RDF model
      conn.add(alice, RDF.TYPE, Engineer);
      conn.add(bob, RDF.TYPE, Doctor);
      conn.add(alice, friendOf, bob);
      conn.commit();

      // prepare a query to run against the virtual model repository
      String queryString =
        "PREFIX ex: <http://example.org/ontology/>\n" +
        "SELECT * WHERE {" +
        "?x ex:friendOf ?y . ?x rdf:type/rdfs:subClassOf* ?xC . ?y rdf:type/
rdfs:subClassOf* ?yC" +
        "} ORDER BY ?x ?xC ?y ?yC\n" ;
        ;
      TupleQuery tupleQuery = vmConn.prepareTupleQuery(QueryLanguage.SPARQL,
queryString);

      // run the query: no results will be returned because nobody is a Person
      try (TupleQueryResult result = tupleQuery.evaluate()) {
        int resultCount = 0;
        while (result.hasNext()) {
          resultCount++;
          BindingSet bindingSet = result.next();
          psOut.println("values of x | xC | y | yC: " +
                  bindingSet.getValue("x") + " | " +
bindingSet.getValue("xC") + " | " +
                  bindingSet.getValue("y") + " | " +
bindingSet.getValue("yC"));
        }
        psOut.println("number of results: " + resultCount);
      }

      // add class hierarchy info to the second model
      conn2.add(Doctor, RDFS.SUBCLASSOF, Person);
      conn2.add(Engineer, RDFS.SUBCLASSOF, Person);
      conn2.commit();

      // run the same query again: returns some additional info in the results
      try (TupleQueryResult result = tupleQuery.evaluate()) {
        int resultCount = 0;
        while (result.hasNext()) {
          resultCount++;
          BindingSet bindingSet = result.next();
          psOut.println("values of x | xC | y | yC: " +
                  bindingSet.getValue("x") + " | " +
bindingSet.getValue("xC") + " | " +
                  bindingSet.getValue("y") + " | " +
bindingSet.getValue("yC"));
        }
```

```
        psOut.println("number of results: " + resultCount);
      }
    }
    finally {
      if (conn != null && conn.isOpen()) {
        conn.clear();
        conn.close();
      }
      OracleUtils.dropSemanticModelAndTables(op.getOracleDB(), model, null, 
null, networkOwner, networkName);
      sr.shutDown();
      store.shutDown();

      if (conn2 != null && conn2.isOpen()) {
          conn2.clear();
          conn2.close();
      }
      OracleUtils.dropSemanticModelAndTables(op.getOracleDB(), model2, null, 
null, networkOwner, networkName);
      sr2.shutDown();
      store2.shutDown();

      vmSr.shutDown();
      vmStore.shutDown();

      op.close();
    }
  }
}
```

To compile this example, execute the following command:

```
javac -classpath $CP SimpleVirtualModel.java
```

To run this example for an existing schema-private network whose owner is SCOTT and name is NET1, execute the following command:

```
java -classpath $CP SimpleVirtualModel jdbc:oracle:thin:@localhost:1521:ORCL 
scott <password-for-scott> TestModel TestOntology TestVM scott net1
```

The expected output of the Java command might appear as follows:

```
values of x | xC | y | yC: http://example.org/people/alice | http://
example.org/ontology/Engineer | http://example.org/people/bob | http://
example.org/ontology/Doctor
number of results: 1
values of x | xC | y | yC: http://example.org/people/alice | http://
example.org/ontology/Person   | http://example.org/people/bob | http://
example.org/ontology/Doctor
values of x | xC | y | yC: http://example.org/people/alice | http://
example.org/ontology/Person   | http://example.org/people/bob | http://
example.org/ontology/Person
values of x | xC | y | yC: http://example.org/people/alice | http://
example.org/ontology/Engineer | http://example.org/people/bob | http://
```

```
example.org/ontology/Doctor
values of x | xC | y | yC: http://example.org/people/alice | http://
example.org/ontology/Engineer | http://example.org/people/bob | http://
example.org/ontology/Person
number of results: 4
```

8.14.16 Example 16: Graph Validation with SHACL

Example 8-20 shows the `ShaclExample.java` file, which loads a data graph into RDF graph `DATA_GRAPH1` and then validates this data against SHACL shape `ex:AgeCountShape`.

Example 8-20 SHACL Validation

```
import java.io.IOException;
import java.io.PrintStream;
import java.io.StringReader;
import java.sql.SQLException;

import org.eclipse.rdf4j.common.exception.ValidationException;
import org.eclipse.rdf4j.model.Model;
import org.eclipse.rdf4j.model.vocabulary.RDF4J;
import org.eclipse.rdf4j.repository.Repository;
import org.eclipse.rdf4j.repository.RepositoryConnection;
import org.eclipse.rdf4j.repository.RepositoryException;
import org.eclipse.rdf4j.repository.sail.SailRepository;
import org.eclipse.rdf4j.repository.sail.SailRepositoryConnection;
import org.eclipse.rdf4j.rio.RDFFormat;
import org.eclipse.rdf4j.rio.RDFParseException;
import org.eclipse.rdf4j.rio.Rio;
import org.eclipse.rdf4j.rio.WriterConfig;
import org.eclipse.rdf4j.rio.helpers.BasicWriterSettings;
import org.eclipse.rdf4j.sail.shacl.ShaclSail;

import oracle.rdf4j.adapter.OracleDB;
import oracle.rdf4j.adapter.OraclePool;
import oracle.rdf4j.adapter.OracleRepository;
import oracle.rdf4j.adapter.OracleSailStore;
import oracle.rdf4j.adapter.shacl.OracleShaclSail;
import oracle.rdf4j.adapter.utils.OracleUtils;

public class ShaclExample {

  public static void main(String[] args) {
    PrintStream psOut = System.out;

    String jdbcUrl = args[0];
    String user = args[1];
    String password = args[2];
    String networkOwner = args[3];
    String networkName = args[4];

    String model = "DATA_GRAPH1";

    OraclePool op = null;
    OracleSailStore store = null;
```

```
Repository sr = null;
RepositoryConnection conn = null;

try {
  op = new OraclePool(jdbcUrl, user, password);
  store = new OracleSailStore(op, model, networkOwner, networkName);

  // load sample data graph
  try {
    sr = new OracleRepository(store);
    conn = sr.getConnection();

    StringReader sampleData = new StringReader(
        String.join(
          "\n", "",
          "@prefix ex: <http://oracle.example.com/ns#> .",

          "ex:Alice a ex:Person .",

          "ex:Bob a ex:Person ;",
          "  ex:age 20 .",

          "ex:Fred a ex:Person ;",
          "  ex:age 30 ;",
          "  ex:age 32 ."
        )
      );

    conn.begin();
    conn.add(sampleData, null, RDFFormat.TURTLE);
    conn.commit();
  }
  catch (IOException e) {
    e.printStackTrace(psOut);
  }
  finally {
    conn.close();
    sr.shutDown();
  }

  // load SHACL shapes graph and validate data
  ShaclSail shaclSail = null;
  SailRepository shaclRepo = null;
  SailRepositoryConnection shaclConn = null;
  try {
    shaclSail = new OracleShaclSail(store);
    shaclRepo = new SailRepository(shaclSail);
    shaclRepo.init();

    shaclConn = shaclRepo.getConnection();

    // add SHACL shapes in a transaction
    // All instances of ex:Person should have exactly one value for ex:age
    StringReader sampleShape = new StringReader(
        String.join(
          "\n", "",
```

```
                "@prefix ex: <http://oracle.example.com/ns#> .",
                "@prefix sh: <http://www.w3.org/ns/shacl#> .",

                "ex:AgeCountShape",
                "  a sh:NodeShape ;",
                "  sh:targetClass ex:Person ;",
                "  sh:property [",
                "    sh:path ex:age ;",
                "    sh:maxCount 1 ;",
                "    sh:minCount 1 ;",
                "  ] ."
              )
            );

          shaclConn.begin();
          // add shape to the reserved named graph http://rdf4j.org/schema/
rdf4j#SHACLShapeGraph
          // clear any existing shapes
          shaclConn.clear(RDF4J.SHACL_SHAPE_GRAPH);
          // add new shapes graph
          shaclConn.add(sampleShape, null, RDFFormat.TURTLE,
RDF4J.SHACL_SHAPE_GRAPH);
          // commit runs bulk validation against the data graph
          shaclConn.commit();
        }
        catch (RDFParseException | IOException e) {
          e.printStackTrace(psOut);
        }
        // SHACL violation will throw a Repository Exception
        catch (RepositoryException e) {
          Throwable cause = e.getCause();
          // Handle validation exception
          if (cause instanceof ValidationException) {
            // Get validation report and print it out
            Model validationReportModel = ((ValidationException)
cause).validationReportAsModel();

            WriterConfig writerConfig = new WriterConfig()
              .set(BasicWriterSettings.INLINE_BLANK_NODES, true)
              .set(BasicWriterSettings.XSD_STRING_TO_PLAIN_LITERAL, true)
              .set(BasicWriterSettings.PRETTY_PRINT, true);

            Rio.write(validationReportModel, psOut, RDFFormat.TURTLE,
writerConfig);
          }
          else {
            e.printStackTrace(psOut);
          }
        }
        finally {
          shaclConn.close();
          shaclRepo.shutDown();
          shaclSail.shutDown();
        }
      }
      catch (SQLException e) {
```

```
        e.printStackTrace(psOut);
      }
      finally {
        if (op != null) {
          try {
            OracleDB oracleDB = op.getOracleDB();
            OracleUtils.dropSemanticModelAndTables(oracleDB, model, null, null, 
networkOwner, networkName);
            op.returnOracleDBtoPool(oracleDB);
          }
          catch (SQLException e) {
            e.printStackTrace(psOut);
          }
        }

        store.shutDown();
        op.close();
      }
    }
}
```

To compile this example, execute the following command:

```
javac -classpath $CP ShaclExample.java
```

To run this example for an existing schema-private network whose owner is SCOTT and name is NET1, execute the following command:

```
java -classpath $CP ShaclExample jdbc:oracle:thin:@localhost:1521:ORCL scott 
<password-for-scott> scott net1
```

The expected output of the Java command might appear as shown. Note that in this case, `ex:Alice` violates the constraint as there is no `ex:age` property, and `ex:Fred` violates the constraint because there are two distinct values for `ex:age`.

```
@prefix dash: <http://datashapes.org/dash#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rsx: <http://rdf4j.org/shacl-extensions#> .
@prefix rdf4j: <http://rdf4j.org/schema/rdf4j#> .
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

[] a sh:ValidationReport;
  sh:conforms false;
  rdf4j:truncated false;
  sh:result [ a sh:ValidationResult;
      sh:focusNode <http://oracle.example.com/ns#Alice>;
      rsx:shapesGraph rdf4j:SHACLShapeGraph;
      sh:resultPath <http://oracle.example.com/ns#age>;
      sh:sourceConstraintComponent sh:MinCountConstraintComponent;
      sh:resultSeverity sh:Violation;
      sh:sourceShape _:c0d709142bb44448ad347272955741bc9
```

```
    ], [ a sh:ValidationResult;
      sh:focusNode <http://oracle.example.com/ns#Fred>;
      rsx:shapesGraph rdf4j:SHACLShapeGraph;
      sh:resultPath <http://oracle.example.com/ns#age>;
      sh:sourceConstraintComponent sh:MaxCountConstraintComponent;
      sh:resultSeverity sh:Violation;
      sh:sourceShape _:c0d709142bb44448ad347272955741bc9
    ] .

_:c0d709142bb44448ad347272955741bc9 a sh:PropertyShape;
  sh:path <http://oracle.example.com/ns#age>;
  sh:minCount 1;
  sh:maxCount 1 .
```

9 User-Defined Inferencing and Querying

RDF graph extension architectures enable the addition of user-defined capabilities.

> **Note:**
>
> User-defined inferencing and querying capabilities is supported only if Oracle JVM is enabled on your Oracle Autonomous Database Serverless deployments. To enable Oracle JVM, see Use Oracle Java in *Using Oracle Autonomous Database Serverless* for more information.

Effective with Oracle Database 12*c* Release 1 (12.1):

- The inference extension architecture enables you to add user-defined inferencing to the presupplied inferencing support.
- The query extension architecture enables you to add user-defined functions and aggregates to be used in SPARQL queries, both through the SEM_MATCH table function and through the support for Apache Jena.

> **Note:**
>
> The capabilities described in this chapter are intended for advanced users. You are assumed to be familiar with the main concepts and techniques described in RDF Graph Overview and OWL Concepts .

- User-Defined Inferencing
  The RDF Graph inference extension architecture enables you to add user-defined inferencing to the presupplied inferencing support.
- User-Defined Functions and Aggregates
  The RDF Graph query extension architecture enables you to add user-defined functions and aggregates to be used in SPARQL queries, both through the SEM_MATCH table function and through the support for Apache Jena.

9.1 User-Defined Inferencing

The RDF Graph inference extension architecture enables you to add user-defined inferencing to the presupplied inferencing support.

- Problem Solved and Benefit Provided by User-Defined Inferencing
- API Support for User-Defined Inferencing
- User-Defined Inference Extension Function Examples

9.1.1 Problem Solved and Benefit Provided by User-Defined Inferencing

Before Oracle Database 12*c* Release 1 (12.1), the Oracle Database inference engine provided native support for OWL 2 RL,RDFS, SKOS, SNOMED (core EL), and user-defined rules, which covered a wide range of applications and requirements. However, there was the limitation that **no new RDF resources** could be created as part of the rules deduction process.

As an example of the capabilities and the limitation before Oracle Database 12*c* Release 1 (12.1), consider the following straightforward inference rule:

```
?C   rdfs:subClassOf  ?D .
?x    rdf:type  ?C  .  ==>  ?x   rdf:type  ?D
```

The preceding rule says that any instance `x` of a subclass `C` will be an instance of `C`'s superclass, `D`. The consequent part of the rule mentions two variables `?x` and `?D`. However, these variables must already exist in the antecedents of the rule, which further implies that these RDF resources must already exist in the knowledge base. In other words, for example, you can derive that `John` is a `Student` only if you know that `John` *exists* as a `GraduateStudent` and if an axiom specifies that the `GraduateStudent` class is a subclass of the `Student` class.

Another example of a limitation is that before Oracle Database 12*c* Release 1 (12.1), the inference functions did not support combining a person's first name and last name to produce a full name as a *new* RDF resource in the inference process. Specifically, this requirement can be captured as a rule like the following:

```
?x   :firstName  ?fn
?x   :lastName   ?ln  ==>  ?x  :fullName  concatenate(?fn ?ln)
```

Effective with Oracle Database 12*c* Release 1 (12.1), the RDF Graph inference extension architecture opens the inference process so that users can implement their own inference extension functions and integrate them into the native inference process. This architecture:

- Supports rules that require the generation of new RDF resources.

  Examples might include concatenation of strings or other string operations, mathematical calculations, and web service callouts.

- Allows implementation of certain existing rules using customized optimizations.

  Although the native OWL inference engine has optimizations for many rules and these rules work efficiently for a variety of large-scale ontologies, for some new untested ontologies a customized optimization of a particular inference component may work even better. In such a case, you can disable a particular inference component in the SEM_APIS.CREATE_INFERRED_GRAPH call and specify a customized inference extension function (using the `inf_ext_user_func_name` parameter) that implements the new optimization.

- Allows the inference engine to be extended with sophisticated inference capabilities.

  Examples might include integrating geospatial reasoning, time interval reasoning, and text analytical functions into the native database inference process.

9.1.2 API Support for User-Defined Inferencing

The primary application programming interface (API) for user-defined inferencing is the SEM_APIS.CREATE_INFERRED_GRAPH procedure, specifically the last parameter:

```
inf_ext_user_func_name  IN VARCHAR2 DEFAULT NULL
```

The `inf_ext_user_func_name` parameter, if specified, identifies one or more user-defined inference functions that implement the specialized logic that you want to use.

- User-Defined Inference Function Requirements

9.1.2.1 User-Defined Inference Function Requirements

Each user-defined inference function that is specified in the `inf_ext_user_func_name` parameter in the call to the SEM_APIS.CREATE_INFERRED_GRAPH procedure must:

- Have a name that starts with the following string: `SEM_INF_`
- Be created with definer's rights, not invoker's rights. (For an explanation of definer's rights and invoker's rights, see *Oracle Database Security Guide*.)

The format of the user-defined inference function must be that shown in the following example for a hypothetical function named `SEM_INF_EXAMPLE`:

```
create or replace function sem_inf_example(
    src_tab_view         in  varchar2,
    resource_id_map_view in  varchar2,
    output_tab           in  varchar2,
    action               in  varchar2,
    num_calls            in  number,
    tplInferredLastRound in  number,
    options              in  varchar2 default null,
    optimization_flag    out number,
    diag_message         out varchar2
    )
return boolean
as
  pragma autonomous_transaction;
begin
  if (action = SDO_SEM_INFERENCE.INF_EXT_ACTION_START) then
    <... preparation work ...>
  end if;
  if (action = SDO_SEM_INFERENCE.INF_EXT_ACTION_RUN) then
    <... actual inference logic ...>
    commit;
  end if;
  if (action = SDO_SEM_INFERENCE.INF_EXT_ACTION_END) then
    <... clean up ...>
  end if;
return true;  -- succeed
end;
/
grant execute on sem_inf_example to <network_owner>;
```

In the user-defined function format, the `optimization_flag` output parameter can specify one or more Oracle-defined names that are associated with numeric values. You can specify one or more of the following:

- `SDO_SEM_INFERENCE.INF_EXT_OPT_FLAG_NONE` indicates that the inference engine should not enable any optimizations for the extension function. (This is the default behavior of the inference engine when the `optimization_flag` parameter is not set.)
- `SDO_SEM_INFERENCE.INF_EXT_OPT_FLAG_ALL_IDS` indicates that all triples/quads inferred by the extension function use only resource IDs. In other words, the `output_tab` table only contains resource IDs (columns `gid`, `sid`, `pid`, and `oid`) and does not contain any lexical values (columns `g`, `s`, `p`, and `o` are all null). Enabling this optimization flag allows the inference engine to skip resource ID lookups.

- SDO_SEM_INFERENCE.INF_EXT_OPT_FLAG_NEWDATA_ONLY indicates that all the triples/quads inferred by the extension function are new and do not already exist in src_tab_view. Enabling this optimization flag allows the inference engine to skip checking for duplicates between the output_tab table and src_tab_view. Note that the src_tab_view contains triples/quads from previous rounds of reasoning, including triples/quads inferred from extension functions.
- SDO_SEM_INFERENCE.INF_EXT_OPT_FLAG_UNIQDATA_ONLY indicates that all the triples/quads inferred by the extension function are unique and do not already exist in the output_tab table. Enabling this optimization flag allows the inference engine to skip checking for duplicates within the output_tab table (for example, no need to check for the same triple inferred twice by an extension function). Note that the output_tab table is empty at the beginning of each round of reasoning for an extension function, so uniqueness of the data must only hold for the current round of reasoning.
- SDO_SEM_INFERENCE.INF_EXT_OPT_FLAG_IGNORE_NULL indicates that the inference engine should ignore an inferred triple or quad if the subject, predicate, or object resource is null. The inference engine considers a resource null if both of its columns in the output_tab table are null (for example, subject is null if the s and sid columns are both null). Enabling this optimization flag allows the inference engine to skip invalid triples/quads in the output_tab table. Note that the inference engine interprets null graph columns (g and gid) as the default graph.

To specify more than one value for the optimization_flag output parameter, use the plus sign (+) to concatenate the values. For example:

```
optimization_flag := SDO_SEM_INFERENCE.INF_EXT_OPT_FLAG_ALL_IDS +
                     SDO_SEM_INFERENCE.INF_EXT_OPT_FLAG_NEWDATA_ONLY +
                     SDO_SEM_INFERENCE.INF_EXT_OPT_FLAG_UNIQDATA_ONLY;
```

For more information about using the optimization_flag output parameter, see Example 3: Optimizing Performance.

9.1.3 User-Defined Inference Extension Function Examples

The following examples demonstrate how to use user-defined inference extension functions to create inferred graphs.

- Example 1: Adding Static Triples, Example 2: Adding Dynamic Triples, and Example 3: Optimizing Performance cover the basics of user-defined inference extensions.

  Example 1: Adding Static Triples and Example 2: Adding Dynamic Triples focus on adding new, inferred triples.

  Example 3: Optimizing Performance focuses on optimizing performance.
- Example 4: Temporal Reasoning (Several Related Examples) and Example 5: Spatial Reasoning demonstrate how to handle special data types efficiently by leveraging native Oracle types and operators.

  Example 4: Temporal Reasoning (Several Related Examples) focuses on the xsd:dateTime data type.

  Example 5: Spatial Reasoning focuses on geospatial data types.
- Example 6: Calling a Web Service makes a web service call to the Oracle Geocoder service.

The first three examples assume that the RDF graph EMPLOYEES exists and contains the following RDF data, displayed in Turtle format:

```
:John   :firstName  "John"  ;
        :lastName   "Smith" .

:Mary   :firstName  "Mary"  ;
        :lastName   "Smith" ;
        :name       "Mary Smith" .

:Alice  :firstName  "Alice" .

:Bob    :firstName  "Bob" ;
        :lastName   "Billow" .
```

For requirements and guidelines for creating user-defined inference extension functions, see API Support for User-Defined Inferencing.

- Example 1: Adding Static Triples
- Example 2: Adding Dynamic Triples
- Example 3: Optimizing Performance
- Example 4: Temporal Reasoning (Several Related Examples)
- Example 5: Spatial Reasoning
- Example 6: Calling a Web Service

9.1.3.1 Example 1: Adding Static Triples

The most basic method to infer new data in a user-defined inference extension function is adding static data. Static data does not depend on any existing data in an RDF graph. This is not a common case for a user-defined inference extension function, but it demonstrates the basics of adding triples to an inferred graph. Inserting static data is more commonly done during the preparation phase (that is, `action='START'`) to expand on the existing ontology.

The following user-defined inference extension function (`sem_inf_static`) adds three static triples to an inferred graph:

```
-- this user-defined rule adds static triples
create or replace function sem_inf_static(
    src_tab_view         in  varchar2,
    resource_id_map_view in  varchar2,
    output_tab           in  varchar2,
    action               in  varchar2,
    num_calls            in  number,
    tplInferredLastRound in  number,
    options              in  varchar2 default null,
    optimization_flag    out number,
    diag_message         out varchar2
    )
return boolean
as
  query varchar2(4000);
  pragma autonomous_transaction;
begin
  if (action = 'RUN') then
    -- generic query we use to insert triples
    query :=
      'insert /*+ parallel append */ into ' || output_tab ||
      ' ( s,  p,  o) VALUES ' ||
      ' (:1, :2, :3) ';

    -- execute the query with different values
```

```
    execute immediate query using
      '<http://example.org/S1>', '<http://example.org/P2>', '"01"';

    execute immediate query using
      '<http://example.org/S2>', '<http://example.org/P2>', '"2"^^xsd:int';

    -- duplicate quad
    execute immediate query using
      '<http://example.org/S2>', '<http://example.org/P2>', '"2"^^xsd:int';

    execute immediate query using
      '<http://example.org/S3>', '<http://example.org/P3>', '"3.0"^^xsd:double';

    -- commit our changes
    commit;
  end if;

  -- return true to indicate success
  return true;
end sem_inf_static;
/
show errors;
```

The `sem_inf_static` function inserts new data by executing a SQL insert query, with `output_tab` as the target table for insertion. The `output_tab` table will only contain triples added by the `sem_inf_static` function during the current call (see the `num_calls` parameter). The inference engine will always call a user-defined inference extension function at least three times, once for each possible value of the action parameter (`'START'`, `'RUN'`, and `'END'`). Because `sem_inf_static` does not need to perform any preparation or cleanup, the function only adds data during the `RUN` phase. The extension function can be called more than once during the `RUN` phase, depending on the data inferred during the current round of reasoning.

Although the `sem_inf_static` function makes no checks for existing triples (to prevent duplicate triples), the inference engine will not generate duplicate triples in the resulting inferred graph. The inference engine will filter out duplicates from the `output_tab` table (the data inserted by the extension function) and from the final inferred graph (the RDF graph or RDF graphs and other inferred data). Setting the appropriate optimization flags (using the `optimization_flag` parameter) will disable this convenience feature and improve performance. (See Example 3: Optimizing Performance for more information about optimization flags.)

Although the table definition for `output_tab` shows a column for graph names, the inference engine will ignore and override all graph names on triples added by extension functions when performing Global Inference (default behavior of SEM_APIS.CREATE_INFERRED_GRAPH) and Named Graph Global Inference (NGGI). To add triples to specific named graphs in a user-defined extension function, use NGLI (Named Graph Local Inference). During NGLI, all triples must belong to a named graph (that is, the `gid` and `g` columns of `output_tab` cannot both be null).

The network owner must have execute privileges on the `sem_inf_static` function to use the function for reasoning. The following example shows how to grant the appropriate privileges on the `sem_inf_static` function and create an inferred graph using the function (along with OWLPRIME inference logic):

```
-- grant appropriate privileges
grant execute on sem_inf_static to RDFUSER;

-- create the inferred graph
begin
```

```
  sem_apis.create_inferred_graph(
    'EMPLOYEES_INF'
  , sem_models('EMPLOYEES')
  , sem_rulebases('OWLPRIME')
  , passes => SEM_APIS.REACH_CLOSURE
  , inf_ext_user_func_name => 'sem_inf_static'
  , network_owner=>'RDFUSER'
  , network_name=>'NET1'
);
end;
/
```

The following example displays the newly entailed data:

```
-- formatting
column s format a23;
column p format a23;
column o format a23;
set linesize 100;

-- show results
select s, p, o from table(SEM_MATCH(
    'select ?s ?p ?o where { ?s ?p ?o } order by ?s ?p ?o'
  , sem_models('EMPLOYEES')
  , sem_rulebases('OWLPRIME')
  , null, null, null
  , 'INF_ONLY=T'
  , network_owner=>'RDFUSER'
  , network_name=>'NET1'));
```

The preceding query returns the three unique static triples added by `sem_inf_static`, with no duplicates:

```
S                      P                      O
---------------------- ---------------------- -----------------------
http://example.org/S1  http://example.org/P2  O1
http://example.org/S2  http://example.org/P2  2
http://example.org/S3  http://example.org/P3  3E0
```

9.1.3.2 Example 2: Adding Dynamic Triples

Adding static data is useful, but it is usually done during the preparation (that is, `action='START'`) phase. Adding *dynamic* data involves looking at existing data in the RDF graph and generating new data based on the existing data. This is the most common case for a user-defined inference extension function.

The following user-defined inference extension function (`sem_inf_dynamic`) concatenates the first and last names of employees to create a new triple that represents the full name.

```
-- this user-defined rule adds static triples
create or replace function sem_inf_dynamic(
    src_tab_view         in  varchar2,
    resource_id_map_view in  varchar2,
    output_tab           in  varchar2,
    action               in  varchar2,
    num_calls            in  number,
    tplInferredLastRound in  number,
    options              in  varchar2 default null,
    optimization_flag    out number,
    diag_message         out varchar2
    )
return boolean
```

```
as
  firstNamePropertyId number;
  lastNamePropertyId  number;
  fullNamePropertyId  number;

  sqlStmt    varchar2(4000);
  insertStmt varchar2(4000);
  pragma autonomous_transaction;
begin
  if (action = 'RUN') then
    -- retrieve ID of resource that already exists in the data (will
    -- throw exception if resource does not exist). These will improve
    -- performance of our SQL queries.
    firstNamePropertyId := sdo_sem_inference.oracle_orardf_res2vid('http://example.org/
firstName');
    lastNamePropertyId  := sdo_sem_inference.oracle_orardf_res2vid('http://example.org/
lastName');
    fullNamePropertyId  := sdo_sem_inference.oracle_orardf_res2vid('http://example.org/
name');

    -- SQL query to find all employees and their first and last names
    sqlStmt :=
      'select ids1.sid employeeId,
              values1.value_name firstName,
              values2.value_name lastName
       from   ' || resource_id_map_view || ' values1,
              ' || resource_id_map_view || ' values2,
              ' || src_tab_view || '         ids1,
              ' || src_tab_view || '         ids2
       where  ids1.sid = ids2.sid
         AND  ids1.pid = ' || to_char(firstNamePropertyId,'TM9') || '
         AND  ids2.pid = ' || to_char(lastNamePropertyId,'TM9')  || '
         AND  ids1.oid = values1.value_id
         AND  ids2.oid = values2.value_id
       /* below ensures we have NEWDATA (a no duplicate optimization flag) */
         AND  not exists
               (select 1
                from   ' || src_tab_view || '
                where  sid = ids1.sid AND
                       pid = ' || to_char(fullNamePropertyId,'TM9') || ')';

    -- create the insert statement that concatenates the first and
    -- last names from our sqlStmt into a new triple.
    insertStmt :=
      'insert /*+ parallel append */
       into ' || output_tab || ' (sid, pid, o)
       select employeeId, ' || to_char(fullNamePropertyId,'TM9') || ', ''"'' ||
firstName || '' '' || lastName ||  ''"''
       from   (' || sqlStmt || ')';

    -- execute the insert statement
    execute immediate insertStmt;

    -- commit our changes
    commit;

    -- set our optimization flags indicating we already checked for
    -- duplicates in the RDF graph (src_tab_view)
    optimization_flag := SDO_SEM_INFERENCE.INF_EXT_OPT_FLAG_NEWDATA_ONLY;
  end if;

  -- return true to indicate success
```

```
  return true;
end sem_inf_dynamic;
/
show errors;
```

The `sem_inf_dynamic` function inserts new data using two main steps. First, the function builds a SQL query that collects all first and last names from the existing data. The `sqlStmt` variable stores this SQL query. Next, the function inserts new triples based on the first and last names it collects, to form a full name for each employee. The `insertStmt` variable stores this SQL query. Note that the `insertStmt` query includes the `sqlStmt` query because it is performing an INSERT with a subquery.

The `sqlStmt` query performs a join across two main views: the resource view (`resource_id_map_view`) and the existing data view (`src_tab_view`). The existing data view contains all existing triples but stores the values of those triples using numeric IDs instead of lexical values. Because the `sqlStmt` query must extract the lexical values of the first and last names of an employee, it joins with the resource view twice (once for the first name and once for the last name).

The `sqlStmt` query contains the `PARALLEL` SQL hint to help improve performance. Parallel execution on a balanced hardware configuration can significantly improve performance. (See Example 3: Optimizing Performance for more information.)

The `insertStmt` query also performs a duplicate check to avoid adding a triple if it already exists in the existing data view (`src_tab_view`). The function indicates it has performed this check by enabling the `INF_EXT_OPT_FLAG_NEWDATA_ONLY` optimization flag. Doing the check inside the extension function improves overall performance of the reasoning. Note that the existing data view does not contain the new triples currently being added by the `sem_inf_dynamic` function, so duplicates may still exist within the `output_tab` table. If the `sem_inf_dynamic` function additionally checked for duplicates within the output_tab table, then it could also enable the `INF_EXT_OPT_FLAG_UNIQUEDATA_ONLY` optimization flag.

Both SQL queries use numeric IDs of RDF resources to perform their joins and inserts. Using IDs instead of lexical values improves the performance of the queries. The `sem_inf_dynamic` function takes advantage of this performance benefit by looking up the IDs of the lexical values it plans to use. In this case, the function looks up three URIs representing the first name, last name, and full name properties. If the `sem_inf_dynamic` function inserted all new triples purely as IDs, then it could enable the `INF_EXT_OPT_FLAG_ALL_IDS` optimization flag. For this example, however, the new triples each contain a single, new, lexical value: the full name of the employee.

To create an inferred graph with the `sem_inf_dynamic` function, grant execution privileges to the network owner, then pass the function name to the SEM_APIS.CREATE_INFERRED_GRAPH procedure, as follows:

```
-- grant appropriate privileges
grant execute on sem_inf_dynamic to RDFUSER;

-- create the inferred graph
begin
  sem_apis.create_inferred_graph(
    'EMPLOYEES_INF'
  , sem_models('EMPLOYEES')
  , sem_rulebases('OWLPRIME')
  , passes => SEM_APIS.REACH_CLOSURE
  , inf_ext_user_func_name => 'sem_inf_dynamic'
  , network_owner=>'RDFUSER'
  , network_name=>'NET1'
);
```

```
end;
/
```

The inferred graph should contain the following two new triples added by `sem_inf_dynamic`:

```
S                        P                        O
------------------------ ------------------------ -----------------------
http://example.org/Bob   http://example.org/name  Bob Billow
http://example.org/John  http://example.org/name  John Smith
```

Note that the `sem_inf_dynamic` function in the preceding example did not infer a full name for Mary Smith, because Mary Smith already had their full name specified in the existing data.

9.1.3.3 Example 3: Optimizing Performance

Several techniques can improve the performance of an inference extension function. One such technique is to use the numeric IDs of resources rather than their lexical values in queries. By only using resource IDs, the extension function avoids having to join with the resource view (`resource_id_map_view`), and this can greatly improve query performance. Inference extension functions can obtain additional performance benefits by also using resource IDs when adding new triples to the `output_tab` table (that is, using only using the `gid`, `sid`, `pid`, and `oid` columns of the `output_tab` table).

The following user-defined inference extension function (`sem_inf_related`) infers a new property, `:possibleRelative`, for employees who share the same last name. The SQL queries for finding such employees use only resource IDs (no lexical values, no joins with the resource view). Additionally, the inference extension function in this example inserts the new triples using only resource IDs, allowing the function to enable the `INF_EXT_OPT_FLAG_ALL_IDS` optimization flag.

```
-- this user-defined rule adds static triples
create or replace function sem_inf_related(
    src_tab_view         in  varchar2,
    resource_id_map_view in  varchar2,
    output_tab           in  varchar2,
    action               in  varchar2,
    num_calls            in  number,
    tplInferredLastRound in  number,
    options              in  varchar2 default null,
    optimization_flag    out number,
    diag_message         out varchar2
    )
return boolean
as
  lastNamePropertyId  number;
  relatedPropertyId   number;

  sqlStmt    varchar2(4000);
  insertStmt varchar2(4000);
  pragma autonomous_transaction;
begin
  if (action = 'RUN') then
    -- retrieve ID of resource that already exists in the data (will
    -- throw exception if resource does not exist).
    lastNamePropertyId := sdo_sem_inference.oracle_orardf_res2vid('http://example.org/
lastName');

    -- retreive ID of resource or generate a new ID if resource does
    -- not already exist
    relatedPropertyId := sdo_sem_inference.oracle_orardf_add_res('http://example.org/
```

```
possibleRelative');

    -- SQL query to find all employees that share a last name
    sqlStmt :=
      'select ids1.sid employeeId,
              ids2.sid relativeId
       from   ' || src_tab_view || '        ids1,
              ' || src_tab_view || '        ids2
       where  ids1.pid = ' || to_char(lastNamePropertyId,'TM9') || '
         AND  ids2.pid = ' || to_char(lastNamePropertyId,'TM9') || '
         AND  ids1.oid  = ids2.oid
       /* avoid employees related to themselves */
         AND  ids1.sid != ids2.sid
       /* below ensures we have NEWDATA (a no duplicate optimization flag) */
         AND  not exists
               (select 1
                from   ' || src_tab_view || '
                where  sid = ids1.sid
                  AND  pid = ' || to_char(relatedPropertyId,'TM9') || '
                  AND  oid = ids2.sid)
       /* below ensures we have UNIQDATA (a no duplicate optimization flag) */
         AND  not exists
               (select 1
                from   ' || output_tab || '
                where  sid = ids1.sid
                  AND  pid = ' || to_char(relatedPropertyId,'TM9') || '
                  AND  oid = ids2.sid)';

    -- create the insert statement that only uses resource IDs
    insertStmt :=
      'insert /*+ parallel append */
       into ' || output_tab || ' (sid, pid, oid)
       select employeeId, ' || to_char(relatedPropertyId,'TM9') || ', relativeId
       from   (' || sqlStmt || ')';

    -- execute the insert statement
    execute immediate insertStmt;

    -- commit our changes
    commit;

    -- set flag indicating our new triples
    --   1) are specified using only IDs
    --   2) produce no duplicates with the RDF graph (src_tab_view)
    --   3) produce no duplicates in the output (output_tab)
    optimization_flag := SDO_SEM_INFERENCE.INF_EXT_OPT_FLAG_ALL_IDS +
                         SDO_SEM_INFERENCE.INF_EXT_OPT_FLAG_NEWDATA_ONLY +
                         SDO_SEM_INFERENCE.INF_EXT_OPT_FLAG_UNIQDATA_ONLY;
  end if;

  -- return true to indicate success
  return true;
end sem_inf_related;
/
show errors;
```

The `sem_inf_related` function has a few key differences from previous examples. First, the `sem_inf_related` function queries purely with resource IDs and inserts new triples using only resource IDs. Because all the added triples in the `output_tab` table only use resource IDs, the function can enable the `INF_EXT_OPT_FLAG_ALL_IDS` optimization flag. For optimal performance, functions should try to use resource IDs over lexical values. However, sometimes this is not possible, as in Example 2: Adding Dynamic Triples, which concatenates lexical values to form a new lexical value. Note that in cases like Example 2: Adding Dynamic Triples, it is usually better to join with the resource view (`resource_id_map_view`) than to embed calls to `oracle_orardf_res2vid` within the SQL query. This is due to the overhead of calling the function for each possible match as opposed to joining with another table.

Another key difference in the `sem_inf_related` function is the use of the `oracle_orardf_add_res` function (compared to `oracle_orardf_res2vid`). Unlike the `res2vid` function, the `add_res` function will add a resource to the resource view (`resource_id_map_view`) if the resource does not already exist. Inference extensions functions should use the `add_res` function if adding the resource to the resource view is not a concern. Calling the function multiple times will not generate duplicate entries in the resource view.

The last main difference is the additional `NOT EXISTS` clause in the SQL query. The first `NOT EXISTS` clause avoids adding any triples that may be duplicates of triples already in the RDF graph or triples inferred by other rules (`src_tab_view`). Checking for these duplicates allows `sem_inf_related` to enable the `INF_EXT_OPT_FLAG_NEWDATA_ONLY` optimization flag. The second `NOT EXISTS` clause avoids adding triples that may be duplicates of triples already added by the `sem_inf_related` function to the output_tab table during the current round of reasoning (see the `num_calls` parameter). Checking for these duplicates allows `sem_inf_related` to enable the `INF_EXT_OPT_FLAG_UNIQDATA_ONLY` optimization flag.

Like the `sem_inf_dynamic` example, `sem_inf_related` example uses a `PARALLEL` SQL query hint in its insert statement. Parallel execution on a balanced hardware configuration can significantly improve performance. For a data-intensive application, a good I/O subsystem is usually a critical component to the performance of the whole system.

To create an inferred graph with the `sem_inf_dynamic` function, grant execution privileges to the network owner, then pass the function name to the SEM_APIS.CREATE_INFERRED_GRAPH procedure, as follows:

```
-- grant appropriate privileges
grant execute on sem_inf_related to RDFUSER;

-- create the inferred graph
begin
  sem_apis.create_inferred_graph(
    'EMPLOYEES_INF'
  , sem_models('EMPLOYEES')
  , sem_rulebases('OWLPRIME')
  , passes => SEM_APIS.REACH_CLOSURE
  , inf_ext_user_func_name => 'sem_inf_related'
  , network_owner=>'RDFUSER'
  , network_name=>'NET1'
);
end;
/
```

The inferred graph should contain the following two new triples added by `sem_inf_related`:

```
S                        P                                    O
------------------------ ------------------------------------ ------------------------
http://example.org/John  http://example.org/possibleRelative  http://example.org/Mary
http://example.org/Mary  http://example.org/possibleRelative  http://example.org/John
```

9.1.3.4 Example 4: Temporal Reasoning (Several Related Examples)

User-defined extension functions enable you to better leverage certain data types (like `xsd:dateTime`) in the triples. For example, with user-defined extension functions, it is possible to infer relationships between triples based on the difference between two `xsd:dateTime` values. The three examples in this section explore two different temporal reasoning rules and how to combine them into one inferred graph. The examples assume the models `EVENT` and `EVENT_ONT` exist and contain the following RDF data:

EVENT_ONT

```
@prefix owl:  <http://www.w3.org/2002/07/owl#> .
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#> .
@prefix :     <http://example.org/event/> .

# we model two types of events
:Meeting      rdfs:subClassOf :Event .
:Presentation rdfs:subClassOf :Event .

# events have topics
:topic        rdfs:domain     :Event .

# events have start and end times
:startTime    rdfs:domain     :Event ;
              rdfs:range      xsd:dateTime .
:endTime      rdfs:domain     :Event ;
              rdfs:range      xsd:dateTime .

# duration (in minutes) of an event
:lengthInMins rdfs:domain      :Event ;
              rdfs:range       xsd:integer .

# overlaps property identifies conflicting events
:overlaps     rdfs:domain      :Event ;
              rdf:type         owl:SymmetricProperty .
:noOverlap    rdfs:domain      :Event ;
              rdf:type         owl:SymmetricProperty .
```

EVENT_TBOX

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix :    <http://example.org/event/> .

:m1 rdf:type   :Meeting ;
    :topic     "Beta1 launch" ;
    :startTime "2012-04-01T09:30:00-05:00"^^xsd:dateTime ;
    :endTime   "2012-04-01T11:00:00-05:00"^^xsd:dateTime .

:m2 rdf:type   :Meeting ;
    :topic     "Standards compliance" ;
    :startTime "2012-04-01T12:30:00-05:00"^^xsd:dateTime ;
    :endTime   "2012-04-01T13:30:00-05:00"^^xsd:dateTime .

:p1 rdf:type   :Presentation ;
    :topic     "OWL Reasoners" ;
    :startTime "2012-04-01T11:00:00-05:00"^^xsd:dateTime ;
    :endTime   "2012-04-01T13:00:00-05:00"^^xsd:dateTime .
```

The examples are as follow.

- Example 4a: Duration Rule
- Example 4b: Overlap Rule

- Example 4c: Duration and Overlap Rules

9.1.3.4.1 Example 4a: Duration Rule

The following user-defined inference extension function (`sem_inf_durations`) infers the duration in minutes of events, given the start and end times of an event. For example, an event starting at 9:30 AM and ending at 11:00 AM has duration of 90 minutes. The following extension function extracts the start and end times for each event, converts the `xsd:dateTime` values into Oracle timestamps, then computes the difference between the timestamps. Notice that this extension function can handle time zones.

```
create or replace function sem_inf_durations(
    src_tab_view         in  varchar2,
    resource_id_map_view in  varchar2,
    output_tab           in  varchar2,
    action               in  varchar2,
    num_calls            in  number,
    tplInferredLastRound in  number,
    options              in  varchar2 default null,
    optimization_flag    out number,
    diag_message         out varchar2
    )
return boolean
as
  eventClassId        number;
  rdfTypePropertyId   number;
  startTimePropertyId number;
  endTimePropertyId   number;
  durationPropertyId  number;

  xsdTimeFormat       varchar2(100);
  sqlStmt             varchar2(4000);
  insertStmt          varchar2(4000);

  pragma autonomous_transaction;
begin
  if (action = 'RUN') then
    -- retrieve ID of resource that already exists in the data (will
    -- throw exception if resource does not exist).
    eventClassId        := sdo_sem_inference.oracle_orardf_res2vid(
                             'http://example.org/event/Event',
                             p_network_owner=>'RDFUSER',
                             p_network_name=>'NET1');
    startTimePropertyId := sdo_sem_inference.oracle_orardf_res2vid(
                             'http://example.org/event/startTime',
                             p_network_owner=>'RDFUSER',
                             p_network_name=>'NET1');
    endTimePropertyId   := sdo_sem_inference.oracle_orardf_res2vid(
                             'http://example.org/event/endTime',
                             p_network_owner=>'RDFUSER',
                             p_network_name=>'NET1');
    durationPropertyId  := sdo_sem_inference.oracle_orardf_res2vid(
                             'http://example.org/event/lengthInMins',
                             p_network_owner=>'RDFUSER',
                             p_network_name=>'NET1');
    rdfTypePropertyId   := sdo_sem_inference.oracle_orardf_res2vid(
                             'http://www.w3.org/1999/02/22-rdf-syntax-ns#type',
                             p_network_owner=>'RDFUSER',
                             p_network_name=>'NET1');

    -- set the TIMESTAMP format we will use to parse XSD times
```

```
    xsdTimeFormat := 'YYYY-MM-DD"T"HH24:MI:SSTZH:TZM';

    -- query we use to extract the event ID and start/end times.
    sqlStmt :=
      'select ids1.sid eventId,
              TO_TIMESTAMP_TZ(values1.value_name,''YYYY-MM-DD"T"HH24:MI:SSTZH:TZM'')
startTime,
              TO_TIMESTAMP_TZ(values2.value_name,''YYYY-MM-DD"T"HH24:MI:SSTZH:TZM'')
endTime
       from   ' || resource_id_map_view || ' values1,
              ' || resource_id_map_view || ' values2,
              ' || src_tab_view || '         ids1,
              ' || src_tab_view || '         ids2,
              ' || src_tab_view || '         ids3
       where  ids1.sid = ids3.sid
         AND  ids3.pid = ' || to_char(rdfTypePropertyId,'TM9') || '
         AND  ids3.oid = ' || to_char(eventClassId,'TM9')      || '
         AND  ids1.sid = ids2.sid
         AND  ids1.pid = ' || to_char(startTimePropertyId,'TM9') || '
         AND  ids2.pid = ' || to_char(endTimePropertyId,'TM9')   || '
         AND  ids1.oid = values1.value_id
         AND  ids2.oid = values2.value_id
       /* ensures we have NEWDATA */
         AND  not exists
               (select 1
                from   ' || src_tab_view || '
                where  sid = ids3.sid
                  AND  pid = ' || to_char(durationPropertyId,'TM9') || ')
       /* ensures we have UNIQDATA */
         AND  not exists
               (select 1
                from   ' || output_tab || '
                where  sid = ids3.sid
                  AND  pid = ' || to_char(durationPropertyId,'TM9') || ')';

    -- compute the difference (in minutes) between the two Oracle
    -- timestamps from our sqlStmt query.  Store the minutes as
    -- xsd:integer.
    insertStmt :=
      'insert /*+ parallel append */ into ' || output_tab || ' (sid, pid, o)
       select eventId,
              ' || to_char(durationPropertyId,'TM9') || ',
              ''"'' || minutes || ''"^^xsd:integer''
       from   (
         select eventId,
                (extract(day    from (endTime - startTime))*24*60 +
                 extract(hour   from (endTime - startTime))*60 +
                 extract(minute from (endTime - startTime))) minutes
         from   (' || sqlStmt || '))';

    -- execute the query
    execute immediate insertStmt;

    -- commit our changes
    commit;
  end if;

  -- we already checked for duplicates in src_tab_view (NEWDATA) and
  -- in output_tab (UNIQDATA)
  optimization_flag := SDO_SEM_INFERENCE.INF_EXT_OPT_FLAG_NEWDATA_ONLY +
                       SDO_SEM_INFERENCE.INF_EXT_OPT_FLAG_UNIQDATA_ONLY;
```

```
  -- return true to indicate success
  return true;

  -- handle any exceptions
  exception
    when others then
      diag_message := 'error occurred: ' || SQLERRM;
      return false;
end sem_inf_durations;
/
show errors;
```

The `sem_inf_durations` function leverages built-in Oracle temporal functions to compute the event durations. First, the function converts the `xsd:dateTime` literal value to an Oracle `TIMESTAMP` object using the `TO_TIMESTAMP_TZ` function. Taking the difference between two Oracle `TIMESTAMP` objects produces an `INTERVAL` object that represents a time interval. Using the `EXTRACT` operator, the `sem_inf_durations` function computes the duration of each event in minutes by extracting the days, hours, and minutes out of the duration intervals.

Because the `sem_inf_durations` function checks for duplicates against both data in the existing model (`src_tab_view`) and data in the `output_tab` table, it can enable the `INF_EXT_OPT_FLAG_NEWDATA_ONLY` and `INF_EXT_OPT_FLAG_UNIQDATA_ONLY` optimization flags. (See Example 3: Optimizing Performance for more information about optimization flags.)

Notice that unlike previous examples, `sem_inf_durations` contains an exception handler. Exception handlers are useful for debugging issues in user-defined inference extension functions. To produce useful debugging messages, catch exceptions in the extension function, set the `diag_message` parameter to reflect the error, and return `FALSE` to indicate that an error occurred during execution of the extension function. The `sem_inf_durations` function catches all exceptions and sets the `diag_message` value to the exception message.

To create an inferred graph with the `sem_inf_durations` function, grant execution privileges to RDFUSER, then pass the function name to the SEM_APIS.CREATE_INFERRED_GRAPH procedure, as follows:

```
-- grant appropriate privileges
grant execute on sem_inf_durations to RDFUSER;

-- create the inferred graph
begin
  sem_apis.create_inferred_graph(
    'EVENT_INF'
  , sem_models('EVENT', 'EVENT_ONT')
  , sem_rulebases('OWLPRIME')
  , passes => SEM_APIS.REACH_CLOSURE
  , inf_ext_user_func_name => 'sem_inf_durations'
  , network_owner=>'RDFUSER'
  , network_name=>'NET1'
);
end;
/
```

In addition to the triples inferred by OWLPRIME, the inferred graph should contain the following three new triples added by `sem_inf_durations`:

```
S                            P                                      O
---------------------------- -------------------------------------- ---------
http://example.org/event/m1  http://example.org/event/lengthInMins  90
http://example.org/event/m2  http://example.org/event/lengthInMins  60
http://example.org/event/p1  http://example.org/event/lengthInMins  120
```

9.1.3.4.2 Example 4b: Overlap Rule

The following user-defined inference extension function (`sem_inf_overlap`) infers whether two events overlap. Two events overlap if one event starts while the other event is in progress. The function extracts the start and end times for every pair of events, converts the `xsd:dateTime` values into Oracle timestamps, then computes whether one event starts within the other.

```
create or replace function sem_inf_overlap(
    src_tab_view         in  varchar2,
    resource_id_map_view in  varchar2,
    output_tab           in  varchar2,
    action               in  varchar2,
    num_calls            in  number,
    tplInferredLastRound in  number,
    options              in  varchar2 default null,
    optimization_flag    out number,
    diag_message         out varchar2
    )
return boolean
as
  eventClassId        number;
  rdfTypePropertyId   number;
  startTimePropertyId number;
  endTimePropertyId   number;
  overlapsPropertyId  number;
  noOverlapPropertyId number;

  xsdTimeFormat       varchar2(100);
  sqlStmt             varchar2(4000);
  insertStmt          varchar2(4000);

  pragma autonomous_transaction;
begin
  if (action = 'RUN') then
    -- retrieve ID of resource that already exists in the data (will
    -- throw exception if resource does not exist).
    eventClassId        := sdo_sem_inference.oracle_orardf_res2vid(
                             'http://example.org/event/Event',
                             p_network_owner=>'RDFUSER',
                             p_network_name=>'NET1');
    startTimePropertyId := sdo_sem_inference.oracle_orardf_res2vid(
                             'http://example.org/event/startTime',
                             p_network_owner=>'RDFUSER',
                             p_network_name=>'NET1');
    endTimePropertyId   := sdo_sem_inference.oracle_orardf_res2vid(
                             'http://example.org/event/endTime',
                             p_network_owner=>'RDFUSER',
                             p_network_name=>'NET1');
    overlapsPropertyId  := sdo_sem_inference.oracle_orardf_res2vid(
                             'http://example.org/event/overlaps',
                             p_network_owner=>'RDFUSER',
                             p_network_name=>'NET1');
    noOverlapPropertyId := sdo_sem_inference.oracle_orardf_res2vid(
                             'http://example.org/event/noOverlap',
                             p_network_owner=>'RDFUSER',
                             p_network_name=>'NET1');
    rdfTypePropertyId   := sdo_sem_inference.oracle_orardf_res2vid(
                             'http://www.w3.org/1999/02/22-rdf-syntax-ns#type',
                             p_network_owner=>'RDFUSER',
                             p_network_name=>'NET1');
```

```
    -- set the TIMESTAMP format we will use to parse XSD times
    xsdTimeFormat := 'YYYY-MM-DD"T"HH24:MI:SSTZH:TZM';

    -- query we use to extract the event ID and start/end times.
    sqlStmt :=
      'select idsA1.sid eventAId,
              idsB1.sid eventBId,
              TO_TIMESTAMP_TZ(valuesA1.value_name,''YYYY-MM-DD"T"HH24:MI:SSTZH:TZM'')
startTimeA,
              TO_TIMESTAMP_TZ(valuesA2.value_name,''YYYY-MM-DD"T"HH24:MI:SSTZH:TZM'')
endTimeA,
              TO_TIMESTAMP_TZ(valuesB1.value_name,''YYYY-MM-DD"T"HH24:MI:SSTZH:TZM'')
startTimeB,
              TO_TIMESTAMP_TZ(valuesB2.value_name,''YYYY-MM-DD"T"HH24:MI:SSTZH:TZM'')
endTimeB
       from   ' || resource_id_map_view || ' valuesA1,
              ' || resource_id_map_view || ' valuesA2,
              ' || resource_id_map_view || ' valuesB1,
              ' || resource_id_map_view || ' valuesB2,
              ' || src_tab_view || '         idsA1,
              ' || src_tab_view || '         idsA2,
              ' || src_tab_view || '         idsA3,
              ' || src_tab_view || '         idsB1,
              ' || src_tab_view || '         idsB2,
              ' || src_tab_view || '         idsB3
       where  idsA1.sid = idsA3.sid
         AND  idsA3.pid = ' || to_char(rdfTypePropertyId,'TM9') || '
         AND  idsA3.oid = ' || to_char(eventClassId,'TM9')      || '
         AND  idsB1.sid = idsB3.sid
         AND  idsB3.pid = ' || to_char(rdfTypePropertyId,'TM9') || '
         AND  idsB3.oid = ' || to_char(eventClassId,'TM9')      || '
       /* only do half the checks, our TBOX ontology will handle symmetries */
         AND  idsA1.sid < idsB1.sid
       /* grab values of startTime and endTime for event A */
         AND  idsA1.sid = idsA2.sid
         AND  idsA1.pid = ' || to_char(startTimePropertyId,'TM9') || '
         AND  idsA2.pid = ' || to_char(endTimePropertyId,'TM9')   || '
         AND  idsA1.oid = valuesA1.value_id
         AND  idsA2.oid = valuesA2.value_id
       /* grab values of startTime and endTime for event B */
         AND  idsB1.sid = idsB2.sid
         AND  idsB1.pid = ' || to_char(startTimePropertyId,'TM9') || '
         AND  idsB2.pid = ' || to_char(endTimePropertyId,'TM9')   || '
         AND  idsB1.oid = valuesB1.value_id
         AND  idsB2.oid = valuesB2.value_id
       /* ensures we have NEWDATA */
         AND  not exists
               (select 1
                from   ' || src_tab_view || '
                where  sid = idsA1.sid
                  AND  oid = idsB1.sid
                  AND  pid in (' || to_char(overlapsPropertyId,'TM9')  || ',' ||
                                    to_char(noOverlapPropertyId,'TM9') || '))
       /* ensures we have UNIQDATA */
         AND  not exists
               (select 1
                from   ' || output_tab   || '
                where  sid = idsA1.sid
                  AND  oid = idsB1.sid
                  AND  pid in (' || to_char(overlapsPropertyId,'TM9')  || ',' ||
                                    to_char(noOverlapPropertyId,'TM9') || '))';
```

```
    -- compare the two event times
    insertStmt :=
      'insert /*+ parallel append */ into ' || output_tab || ' (sid, pid, oid)
       select eventAId, overlapStatusId, eventBId
       from    (
         select eventAId,
                (case
                 when (startTimeA < endTimeB and
                       startTimeA > startTimeB) then
                   ' || to_char(overlapsPropertyId,'TM9') || '
                 when (startTimeB < endTimeA and
                       startTimeB > startTimeA) then
                   ' || to_char(overlapsPropertyId,'TM9') || '
                 else
                   ' || to_char(noOverlapPropertyId,'TM9') || '
                 end) overlapStatusId,
                 eventBId
         from    (' || sqlStmt || '))';

    -- execute the query
    execute immediate insertStmt;

    -- commit our changes
    commit;
  end if;

  -- we only use ID values in the output_tab and we check for
  -- duplicates with our NOT EXISTS clause.
  optimization_flag := SDO_SEM_INFERENCE.INF_EXT_OPT_FLAG_ALL_IDS +
                       SDO_SEM_INFERENCE.INF_EXT_OPT_FLAG_NEWDATA_ONLY +
                       SDO_SEM_INFERENCE.INF_EXT_OPT_FLAG_UNIQDATA_ONLY;

  -- return true to indicate success
  return true;

  -- handle any exceptions
  exception
    when others then
      diag_message := 'error occurred: ' || SQLERRM;
      return false;
end sem_inf_overlap;
/
show errors;
```

The `sem_inf_overlap` function is similar to the `sem_inf_durations` function in Example 4b: Overlap Rule. The main difference between the two is that the query in `sem_inf_overlap` contains more joins and enables the INF_EXT_OPT_FLAG_ALL_IDS optimization flag because it does not need to generate new lexical values. (See Example 3: Optimizing Performance for more information about optimization flags.)

To create an inferred graph with the `sem_inf_overlap` function, grant execution privileges to RDFUSER, then pass the function name to the SEM_APIS.CREATE_INFERRED_GRAPH procedure, as follows:

```
-- grant appropriate privileges
grant execute on sem_inf_overlap to RDFUSER;

-- create the inferred graph
begin
  sem_apis.create_inferred_graph(
    'EVENT_INF'
  , sem_models('EVENT', 'EVENT_ONT')
```

```
  , sem_rulebases('OWLPRIME')
  , passes => SEM_APIS.REACH_CLOSURE
  , inf_ext_user_func_name => 'sem_inf_overlap'
  , network_owner=>'RDFUSER'
  , network_name=>'NET1'
);
end;
/
```

In addition to the triples inferred by OWLPRIME, the inferred graph should contain the following six new triples added by `sem_inf_overlap`:

```
S                            P                                   O
---------------------------- -----------------------------------
----------------------------
http://example.org/event/m1  http://example.org/event/noOverlap  http://example.org/
event/m2
http://example.org/event/m1  http://example.org/event/noOverlap  http://example.org/
event/p1
http://example.org/event/m2  http://example.org/event/noOverlap  http://example.org/
event/m1
http://example.org/event/m2  http://example.org/event/overlaps   http://example.org/
event/p1
http://example.org/event/p1  http://example.org/event/noOverlap  http://example.org/
event/m1
http://example.org/event/p1  http://example.org/event/overlaps   http://example.org/
event/m2
```

9.1.3.4.3 Example 4c: Duration and Overlap Rules

The example in this section uses the extension functions from Example 4a: Duration Rule (`sem_inf_durations`) and Example 4b: Overlap Rule (`sem_inf_overlap`) together to produce a single inferred graph. The extension functions are left unmodified for this example.

To create an inferred graph using multiple extension functions, use a comma to separate each extension function passed to the `inf_ext_user_func_name` parameter of SEM_APIS.CREATE_INFERRED_GRAPH. The following example assumes that the RDFUSER has already been granted the appropriate privileges on the extension functions.

```
-- use multiple user-defined inference functions
begin
  sem_apis.create_inferred_graph(
    'EVENT_INF'
  , sem_models('EVENT', 'EVENT_ONT')
  , sem_rulebases('OWLPRIME')
  , passes => SEM_APIS.REACH_CLOSURE
  , inf_ext_user_func_name => 'sem_inf_durations,sem_inf_overlap'
  , network_owner=>'RDFUSER'
  , network_name=>'NET1'
);
end;
/
```

In addition to the triples inferred by OWLPRIME, the inferred graph should contain the following nine new triples added by `sem_inf_durations` and `sem_inf_overlap`:

```
S                            P                                      O
---------------------------- --------------------------------------
----------------------------
http://example.org/event/m1  http://example.org/event/lengthInMins  90
http://example.org/event/m1  http://example.org/event/noOverlap     http://example.org/
```

```
event/m2
http://example.org/event/m1  http://example.org/event/noOverlap     http://example.org/
event/p1
http://example.org/event/m2  http://example.org/event/lengthInMins  60
http://example.org/event/m2  http://example.org/event/noOverlap     http://example.org/
event/m1
http://example.org/event/m2  http://example.org/event/overlaps      http://example.org/
event/p1
http://example.org/event/p1  http://example.org/event/lengthInMins  120
http://example.org/event/p1  http://example.org/event/noOverlap     http://example.org/
event/m1
http://example.org/event/p1  http://example.org/event/overlaps      http://example.org/
event/m2
```

Notice that the extension functions, `sem_inf_durations` and `sem_inf_overlap`, did not need to use the same optimization flags. It is possible to use extension functions with contradictory optimization flags (for example, one function using `INF_EXT_OPT_FLAG_ALL_IDS` and another function inserting all new triples as lexical values).

9.1.3.5 Example 5: Spatial Reasoning

User-defined inference extension functions can also leverage geospatial data types, like WKT (well-known text), to perform spatial reasoning. For example, with user-defined extension functions, it is possible to infer a "contains" relationship between geometric entities, such as states and cities.

The example in this section demonstrates how to infer whether a geometry (a US state) contains a point (a US city). This example assumes the RDF network already has a spatial index (described in section 1.6.6.2). This example also assumes the RDF graph `STATES` exists and contains the following RDF data:

```
@prefix orageo: <http://xmlns.oracle.com/rdf/geo/> .
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs:   <http://www.w3.org/2000/01/rdf-schema#> .
@prefix :       <http://example.org/geo/> .

:Colorado rdf:type  :State ;
          :boundary "Polygon((-109.0448 37.0004, -102.0424 36.9949, -102.0534 41.0006,
-109.0489 40.9996, -109.0448 37.0004))"^^orageo:WKTLiteral .
:Utah     rdf:type  :State ;
          :boundary "Polygon((-114.0491 36.9982, -109.0462 37.0026, -109.0503 40.9986,
-111.0471 41.0006, -111.0498 41.9993, -114.0395 41.9901, -114.0491
36.9982))"^^orageo:WKTLiteral .
:Wyoming  rdf:type  :State ;
          :boundary "Polygon((-104.0556 41.0037, -104.0584 44.9949, -111.0539 44.9998,
-111.0457 40.9986, -104.0556 41.0037))"^^orageo:WKTLiteral

:StateCapital rdfs:subClassOf :City ;

:Denver   rdf:type  :StateCapital ;
          :location "Point(-104.984722 39.739167)"^^orageo:WKTLiteral .
:SaltLake rdf:type  :StateCaptial ;
          :location "Point(-111.883333 40.75)"^^orageo:WKTLiteral .
:Cheyenne rdf:type  :StateCapital ;
          :location "Point(-104.801944 41.145556)"^^orageo:WKTLiteral .
```

The following user-defined inference extension function (`sem_inf_capitals`) searches for capital cities within each state using the WKT geometries. If the function finds a capital city, it infers the city is the capital of the state containing it.

```
create or replace function sem_inf_capitals(
    src_tab_view         in  varchar2,
    resource_id_map_view in  varchar2,
    output_tab           in  varchar2,
    action               in  varchar2,
    num_calls            in  number,
    tplInferredLastRound in  number,
    options              in  varchar2 default null,
    optimization_flag    out number,
    diag_message         out varchar2
    )
return boolean
as
  stateClassId        number;
  capitalClassId      number;

  boundaryPropertyId  number;
  locationPropertyId  number;
  rdfTypePropertyId   number;
  capitalPropertyId   number;

  defaultSRID         number := 8307;

  xsdTimeFormat       varchar2(100);
  sqlStmt             varchar2(4000);
  insertStmt          varchar2(4000);

  pragma autonomous_transaction;
begin
  if (action = 'RUN') then
    -- retrieve ID of resource that already exists in the data (will
    -- throw exception if resource does not exist).
    stateClassId       := sdo_sem_inference.oracle_orardf_res2vid(
                            'http://example.org/geo/State',
                            p_network_owner=>'RDFUSER',
                            p_network_name=>'NET1');
    capitalClassId     := sdo_sem_inference.oracle_orardf_res2vid(
                            'http://example.org/geo/StateCapital',
                            p_network_owner=>'RDFUSER',
                            p_network_name=>'NET1');
    boundaryPropertyId := sdo_sem_inference.oracle_orardf_res2vid(
                            'http://example.org/geo/boundary',
                            p_network_owner=>'RDFUSER',
                            p_network_name=>'NET1');
    locationPropertyId := sdo_sem_inference.oracle_orardf_res2vid(
                            'http://example.org/geo/location',
                            p_network_owner=>'RDFUSER',
                            p_network_name=>'NET1');
    rdfTypePropertyId  := sdo_sem_inference.oracle_orardf_res2vid(
                            'http://www.w3.org/1999/02/22-rdf-syntax-ns#type',
                            p_network_owner=>'RDFUSER',
                            p_network_name=>'NET1');

    -- retreive ID of resource or generate a new ID if resource does
    -- not already exist
    capitalPropertyId := sdo_sem_inference.oracle_orardf_add_res(
                           'http://example.org/geo/capital',
                           p_network_owner=>'RDFUSER',
                           p_network_name=>'NET1');

    -- query we use to extract the capital cities contained within state boundaries
    sqlStmt :=
```

```
  'select idsA1.sid stateId,
          idsB1.sid cityId
   from   ' || resource_id_map_view || ' valuesA,
          ' || resource_id_map_view || ' valuesB,
          ' || src_tab_view || '         idsA1,
          ' || src_tab_view || '         idsA2,
          ' || src_tab_view || '         idsB1,
          ' || src_tab_view || '         idsB2
   where  idsA1.pid = ' || to_char(rdfTypePropertyId,'TM9') || '
     AND  idsA1.oid = ' || to_char(stateClassId,'TM9')      || '
     AND  idsB1.pid = ' || to_char(rdfTypePropertyId,'TM9') || '
     AND  idsB1.oid = ' || to_char(capitalClassId,'TM9')    || '
   /* grab geometric lexical values */
     AND  idsA2.sid = idsA1.sid
     AND  idsA2.pid = ' || to_char(boundaryPropertyId,'TM9')|| '
     AND  idsA2.oid = valuesA.value_id
     AND  idsB2.sid = idsB1.sid
     AND  idsB2.pid = ' || to_char(locationPropertyId,'TM9')|| '
     AND  idsB2.oid = valuesB.value_id
   /* compare geometries to see if city is contained by state */
     AND  SDO_RELATE(
            SDO_RDF.getV$GeometryVal(
              valuesA.value_type,
              valuesA.vname_prefix,
              valuesA.vname_suffix,
              valuesA.literal_type,
              valuesA.language_type,
              valuesA.long_value,
              ' || to_char(defaultSRID,'TM9') || '),
            SDO_RDF.getV$GeometryVal(
              valuesB.value_type,
              valuesB.vname_prefix,
              valuesB.vname_suffix,
              valuesB.literal_type,
              valuesB.language_type,
              valuesB.long_value,
              ' || to_char(defaultSRID,'TM9') || '),
            ''mask=CONTAINS'') = ''TRUE''
   /* ensures we have NEWDATA and only check capitals not assigned to a state */
     AND  not exists
           (select 1
            from   ' || src_tab_view || '
            where  pid = ' || to_char(capitalPropertyId,'TM9') || '
              AND  (sid = idsA1.sid OR oid = idsB1.sid))
   /* ensures we have UNIQDATA and only check capitals not assigned to a state */
     AND  not exists
           (select 1
            from   ' || output_tab   || '
            where  pid = ' || to_char(capitalPropertyId,'TM9') || '
              AND  (sid = idsA1.sid OR oid = idsB1.sid))';

-- insert new triples using only IDs
insertStmt :=
  'insert /*+ parallel append */ into ' || output_tab || ' (sid, pid, oid)
   select stateId, ' || to_char(capitalPropertyId,'TM9') || ', cityId
   from   (' || sqlStmt || ')';

-- execute the query
execute immediate insertStmt;

-- commit our changes
commit;
```

```
  end if;

  -- we only use ID values in the output_tab and we check for
  -- duplicates with our NOT EXISTS clauses.
  optimization_flag := SDO_SEM_INFERENCE.INF_EXT_OPT_FLAG_ALL_IDS +
                       SDO_SEM_INFERENCE.INF_EXT_OPT_FLAG_NEWDATA_ONLY +
                       SDO_SEM_INFERENCE.INF_EXT_OPT_FLAG_UNIQDATA_ONLY;

  -- return true to indicate success
  return true;

  -- handle any exceptions
  exception
    when others then
      diag_message := 'error occurred: ' || SQLERRM;
      return false;
end sem_inf_capitals;
/
show errors;
```

The `sem_inf_capitals` function is similar to the `sem_inf_durations` function in Example 4a: Duration Rule, in that both functions must convert the lexical values of some triples into Oracle types to leverage native Oracle operators. In the case of `sem_inf_capitals`, the function converts the WKT lexical values encoding polygons and points into the Oracle Spatial SDO_GEOMETRY type, using the `SDO_RDF.getV$GeometryVal` function. The `getV$GeometryVal` function requires arguments mostly provided by the resource view (`resource_id_map_view`) and an additional argument, an ID to a spatial reference system (SRID). The `getV$GeometryVal` function will convert the geometry into the spatial reference system specified by SRID. The `sem_inf_capitals` function uses the default Oracle Spatial reference system, WGS84 Longitude-Latitude, specified by SRID value 8307. (For more information about support in RDF Graph for spatial references systems, see Spatial Support.)

After converting the WKT values into SDO_GEOMETRY types using the `getV$GeometryVal` function, the `sem_inf_capitals` function compares the state geometry with the city geometry to see if the state contains the city. The `SDO_RELATE` operator performs this comparison and returns the literal value `'TRUE'` when the state contains the city. The `SDO_RELATE` operator can perform various different types of comparisons. (See *Oracle Spatial Developer's Guide* for more information about `SDO_RELATE` and other spatial operators.)

To create an inferred graph with the `sem_inf_capitals` function, grant execution privileges to the RDFUSER, then pass the function name to the SEM_APIS.CREATE_INFERRED_GRAPH procedure, as follows:

```
-- grant appropriate privileges
grant execute on sem_inf_capitals to RDFUSER;

-- create the inferred graph
begin
  sem_apis.create_inferred_graph(
    'STATES_INF'
  , sem_models('STATES')
  , sem_rulebases('OWLPRIME')
  , passes => SEM_APIS.REACH_CLOSURE
  , inf_ext_user_func_name => 'sem_inf_capitals'
  , network_owner=>'RDFUSER'
  , network_name=>'NET1'
);
end;
/
```

In addition to the triples inferred by OWLPRIME, the inferred graph should contain the following three new triples added by `sem_inf_capitals`:

```
S                                P                                O
-------------------------------- --------------------------------
--------------------------------
http://example.org/geo/Colorado  http://example.org/geo/capital  http://example.org/geo/
Denver
http://example.org/geo/Utah      http://example.org/geo/capital  http://example.org/geo/
SaltLake
http://example.org/geo/Wyoming   http://example.org/geo/capital  http://example.org/geo/
Cheyenne
```

9.1.3.6 Example 6: Calling a Web Service

This section contains a user-defined inference extension function (`sem_inf_geocoding`) and a related helper procedure (`geocoding`), which enable you to make a web service call to the Oracle Geocoder service. The user-defined inference extension function looks for the object values of triples using predicate `<urn:streetAddress>`, makes callouts to the Oracle public Geocoder service endpoint at `http://maps.oracle.com/geocoder/gcserver`, and inserts the longitude and latitude information as two separate triples.

For example, assume that the RDF graph contains the following assertion:

```
<urn:NEDC>  <urn:streetAddress>  "1 Oracle Dr., Nashua, NH"
```

In this case, an inference call using `sem_inf_geocoding` will produce the following new assertions:

```
<urn:NEDC> <http://www.w3.org/2003/01/geo/wgs84_pos#long>  "-71.46421"
<urn:NEDC> <http://www.w3.org/2003/01/geo/wgs84_pos#lat>   "42.75836"
<urn:NEDC> <http://www.opengis.net/geosparql#asWKT>  "POINT(-71.46421
42.75836)"^^<http://www.opengis.net/geosparql#wktLiteral>
<urn:NEDC> <http://xmlns.oracle.com/rdf/geo/asWKT>   "POINT(-71.46421
42.75836)"^^<http://xmlns.oracle.com/rdf/geo/WKTLiteral>
```

The `sem_inf_geocoding` function is defined as follows:

```
create or replace function sem_inf_geocoding(
    src_tab_view         in  varchar2,
    resource_id_map_view in  varchar2,
    output_tab           in  varchar2,
    action               in  varchar2,
    num_calls            in  number,
    tplInferredLastRound in  number,
    options              in  varchar2 default null,
    optimization_flag    out number,
    diag_message         out varchar2
    )
return boolean
as
  pragma autonomous_transaction;
  iCount integer;

  nLong number;
  nLat  number;
  nWKT  number;
  nOWKT number;
  nStreetAddr number;

  sidTab    dbms_sql.number_table;
```

```
  oidTab    dbms_sql.number_table;

  vcRequestBody varchar2(32767);
  vcStmt        varchar2(32767);
  vcStreeAddr   varchar2(3000);

  type cur_type is ref cursor;
  cursorFind    cur_type;
  vcLong varchar2(100);
  vcLat  varchar2(100);
begin
  if (action = 'START') then
    nLat := sdo_sem_inference.oracle_orardf_add_res(
              'http://www.w3.org/2003/01/geo/wgs84_pos#lat',
              p_network_owner=>'RDFUSER',
              p_network_name=>'NET1');
    nLong := sdo_sem_inference.oracle_orardf_add_res(
               'http://www.w3.org/2003/01/geo/wgs84_pos#long',
               p_network_owner=>'RDFUSER',
               p_network_name=>'NET1');
    nWKT  := sdo_sem_inference.oracle_orardf_add_res(
               'http://www.opengis.net/geosparql#asWKT',
               p_network_owner=>'RDFUSER',
               p_network_name=>'NET1');
    nOWKT := sdo_sem_inference.oracle_orardf_add_res(
               'http://xmlns.oracle.com/rdf/geo/asWKT',
               p_network_owner=>'RDFUSER',
               p_network_name=>'NET1');
  end if;

  if (action = 'RUN') then
    nStreetAddr := sdo_sem_inference.oracle_orardf_res2vid(
                     '<urn:streetAddress>',
                     p_network_owner=>'RDFUSER',
                     p_network_name=>'NET1');
    nLat := sdo_sem_inference.oracle_orardf_res2vid(
              'http://www.w3.org/2003/01/geo/wgs84_pos#lat',
              p_network_owner=>'RDFUSER',
              p_network_name=>'NET1');
    nLong := sdo_sem_inference.oracle_orardf_res2vid(
               'http://www.w3.org/2003/01/geo/wgs84_pos#long',
               p_network_owner=>'RDFUSER',
               p_network_name=>'NET1');
    nWKT  := sdo_sem_inference.oracle_orardf_res2vid(
               'http://www.opengis.net/geosparql#asWKT',
               p_network_owner=>'RDFUSER',
               p_network_name=>'NET1');
    nOWKT := sdo_sem_inference.oracle_orardf_res2vid(
               'http://xmlns.oracle.com/rdf/geo/asWKT',
               p_network_owner=>'RDFUSER',
               p_network_name=>'NET1');

    vcStmt := '
      select /*+ parallel */ distinct s1.sid as s_id, s1.oid as o_id
        from ' || src_tab_view || ' s1
       where s1.pid = :1
         and not exists ( select 1
                            from   ' || src_tab_view || ' x
                           where  x.sid = s1.sid
                             and  x.pid = :2
                        ) ';
    open cursorFind for vcStmt using nStreetAddr, nLong;
```

```
    loop
      fetch cursorFind bulk collect into sidTab, oidTab limit 10000;
      for i in 1..sidTab.count loop
        vcStreeAddr := sdo_sem_inference.oracle_orardf_vid2lit(
                         oidTab(i),
                         p_network_owner=>'RDFUSER',
                         p_network_name=>'NET1');
        -- dbms_output.put_line('Now processing street addr ' || vcStreeAddr);
        geocoding(vcStreeAddr, vcLong, vcLat);
        execute immediate 'insert into ' || output_tab || '(sid,pid,oid,gid,s,p,o,g)
            values(:1, :2, null, null, null, null, :3, null) '
            using sidTab(i), nLong, '"'||vcLong||'"';
        execute immediate 'insert into ' || output_tab || '(sid,pid,oid,gid,s,p,o,g)
            values(:1, :2, null, null, null, null, :3, null) '
            using sidTab(i), nLat, '"'||vcLat||'"';
        execute immediate 'insert into ' || output_tab || '(sid,pid,oid,gid,s,p,o,g)
            values(:1, :2, null, null, null, null, :3, null) '
            using sidTab(i), nWKT, '"POINT('|| vcLong || ' ' ||vcLat ||')"^^<http://
www.opengis.net/geosparql#wktLiteral>';
        execute immediate 'insert into ' || output_tab || '(sid,pid,oid,gid,s,p,o,g)
            values(:1, :2, null, null, null, null, :3, null) '
            using sidTab(i), nOWKT, '"POINT('|| vcLong || ' ' ||vcLat ||')"^^<http://
xmlns.oracle.com/rdf/geo/WKTLiteral>';
      end loop;
      exit when cursorFind%notfound;
    end loop;
    commit;
  end if;
  return true;
end;
/
grant execute on sem_inf_geocoding to RDFUSER;
```

The `sem_inf_geocoding` function makes use of the following helper procedure named `geocoding`, which does the actual HTTP communication with the Geocoder web service endpoint. Note that proper privileges are required to connect to the web server.

```
create or replace procedure geocoding(addr varchar2,
                                      vcLong out varchar2,
                                      vcLat  out varchar2
                                     )
as
  httpReq  utl_http.req;
  httpResp utl_http.resp;

  vcRequestBody varchar2(32767);

  vcBuffer  varchar2(32767);
  idxLat integer;
  idxLatEnd integer;
begin
  vcRequestBody := utl_url.escape('xml_request=<?xml version="1.0" standalone="yes"?>
    <geocode_request vendor="elocation">
      <address_list>
      <input_location id="27010">
      <input_address match_mode="relax_street_type">
        <unformatted country="US">
           <address_line value="'|| addr ||'"/>
        </unformatted>
       </input_address>
      </input_location>
```

```
    </address_list>
    </geocode_request>
  ');
  dbms_output.put_line('request ' || vcRequestBody);

  -- utl_http.set_proxy('<your_proxy_here_if_necessary>', null);
  httpReq := utl_http.begin_request (
    'http://maps.oracle.com/geocoder/gcserver', 'POST');

  utl_http.set_header(httpReq, 'Content-Type', 'application/x-www-form-urlencoded');
  utl_http.set_header(httpReq, 'Content-Length', lengthb(vcRequestBody));

  utl_http.write_text(httpReq, vcRequestBody);

  httpResp := utl_http.get_response(httpReq);

  utl_http.read_text(httpResp, vcBuffer, 32767);
  utl_http.end_response(httpResp);

  -- dbms_output.put_line('response ' || vcBuffer);
  -- Here we are doing some simple string parsing out of an XML.
  -- It is more robust to use XML functions instead.
  idxLat := instr(vcBuffer, 'longitude="');
  idxLatEnd := instr(vcBuffer, '"', idxLat + 12);
  vcLong := substr(vcBuffer, idxLat + 11, idxLatEnd - idxLat - 11);
  dbms_output.put_line('long = ' || vcLong);

  idxLat := instr(vcBuffer, 'latitude="');
  idxLatEnd := instr(vcBuffer, '"', idxLat + 11);
  vcLat := substr(vcBuffer, idxLat + 10, idxLatEnd - idxLat - 10);
  dbms_output.put_line('lat = ' || vcLat);
exception
  when others then
    dbms_output.put_line('geocoding: error ' || dbms_utility.format_error_backtrace || '
'
                                            || dbms_utility.format_error_stack);
end;
/
```

9.2 User-Defined Functions and Aggregates

The RDF Graph query extension architecture enables you to add user-defined functions and aggregates to be used in SPARQL queries, both through the SEM_MATCH table function and through the support for Apache Jena.

The SPARQL 1.1 Standard provides several functions used mainly for filtering and categorizing data obtained by a query. However, you may need specialized functions not supported by the standard.

Some simple examples include finding values that belong to a specific type, or obtaining values with a square sum value that is greater than a certain threshold. Although this can be done by means of combining functions, it may be useful to have a single function that handles the calculations, which also allows for a simpler and shorter query.

The RDF Graph query extension allows you to include your own query functions and aggregates. This architecture allows:

- Custom query functions that can be used just like built-in SPARQL query functions, as explained in API Support for User-Defined Functions

- Custom aggregates that can be used just like built-in SPARQL aggregates, as explained in API Support for User-Defined Aggregates
- Data Types for User-Defined Functions and Aggregates
- API Support for User-Defined Functions
- API Support for User-Defined Aggregates

9.2.1 Data Types for User-Defined Functions and Aggregates

The SDO_RDF_TERM object type is used to represent an RDF term when creating user-defined functions and aggregates.

SDO_RDF_TERM has the following attributes, which correspond to columns in the RDF_VALUE$ table (see Table 1-5 in Statements for a description of these attributes). The CTX1 and FLAGS attributes are reserved for future use and do not have corresponding columns in RDF_VALUE$.

```
SDO_RDF_TERM(
  VALUE_TYPE    VARCHAR2(10),
  VALUE_NAME    CLOB,
  VNAME_PREFIX  CLOB,
  VNAME_SUFFIX  VARCHAR2(512),
  LITERAL_TYPE  VARCHAR2(1000),
  LANGUAGE_TYPE VARCHAR2(80),
  LONG_VALUE    CLOB,
  CTX1          VARCHAR2(4000),
  FLAGS         INTEGER )
```

The following constructors are available for creating SDO_RDF_TERM objects. The first constructor populates each attribute from a single, lexical RDF term string. The second, third, and fourth constructors receive individual attribute values as input. Only the first RDF term string constructor sets values for VNAME_PREFIX and VNAME_SUFFIX. These values are initialized to null by the other constructors.

```
SDO_RDF_TERM (
  rdf_term_str  VARCHAR2)
  RETURN SELF;

SDO_RDF_TERM (
  value_type    VARCHAR2,
  value_name    VARCHAR2,
  literal_type  VARCHAR2,
  language_type VARCHAR2,
  long_value    CLOB)
  RETURN SELF;

SDO_RDF_TERM (
  value_type    VARCHAR2,
  value_name    VARCHAR2,
  literal_type  VARCHAR2,
  language_type VARCHAR2,
  long_value    CLOB,
  ctx1          VARCHAR2)
  RETURN SELF;

SDO_RDF_TERM (
  value_type    VARCHAR2,
  value_name    VARCHAR2,
  literal_type  VARCHAR2,
  language_type VARCHAR2,
```

```
  long_value    CLOB,
  ctx1          VARCHAR2,
  flags         INTEGER)
  RETURN SELF;
```

The SDO_RDF_TERM_LIST type is used to hold a list of SDO_RDF_TERM objects and is defined as `VARRAY(32767) of SDO_RDF_TERM`.

9.2.2 API Support for User-Defined Functions

A user-defined function is created by implementing a PL/SQL function with a specific signature, and a specific URI is used to invoke the function in a SPARQL query pattern.

After each successful inference extension function call, a commit is executed to persist changes made in the inference extension function call. If an inference extension function is defined as autonomous by specifying `pragma autonomous_transaction`, then it should either commit or roll back at the end of its implementation logic. Note that the inference engine may call an extension function multiple times when creating an inferred graph (once per round). Commits and rollbacks from one call will not affect other calls.

- PL/SQL Function Implementation
- Invoking User-Defined Functions from a SPARQL Query Pattern
- User-Defined Function Examples

9.2.2.1 PL/SQL Function Implementation

Each user-defined function must be implemented by a PL/SQL function with a signature in the following format:

```
FUNCTION user_function_name (params IN SDO_RDF_TERM_LIST)
  RETURN SDO_RDF_TERM
```

This signature supports an arbitrary number of RDF term arguments, which are passed in using a single SDO_RDF_TERM_LIST object, and returns a single RDF term as output, which is represented as a single SDO_RDF_TERM object. Type checking or other verifications for these parameters are not performed. You should take steps to validate the data according to the function goals.

Note that PL/SQL supports callouts to functions written in other programming languages, such as C and Java, so the PL/SQL function that implements a user-defined query function can serve only as a wrapper for functions written in other programming languages.

9.2.2.2 Invoking User-Defined Functions from a SPARQL Query Pattern

After a user-defined function is implemented in PL/SQL, it can be invoked from a SPARQL query pattern using a function URI constructed from the prefix `<http://xmlns.oracle.com/rdf/extensions/>` followed by *`schema.package_name.function_name`* if the corresponding PL/SQL function is part of a PL/SQL package, or *`schema.function_name`* if the function is not part of a PL/SQL package. The following are two example function URIs:

```
<http://xmlns.oracle.com/rdf/extensions/my_schema.my_package.my_function>(arg_1, …,
arg_n)

<http://xmlns.oracle.com/rdf/extensions/my_schema.my_function>(arg_1, …, arg_n)
```

9.2.2.3 User-Defined Function Examples

This section presents examples of the implementation of a user-defined function and the use of that function in a FILTER clause, in a SELECT expression, and in a BIND operation.

For the examples, assume that the following data, presented here in N-triple format, exists inside a model called `MYMODEL`:

```
<a>  <p>  "1.0"^^xsd:double .
<b>  <p>  "1.5"^^xsd:float .
<c>  <p>  "3"^^xsd:decimal .
<d>  <p>  "4"^^xsd:string .
```

Example 9-1   User-Defined Function to Calculate Sum of Two Squares

Example 9-1 shows the implementation of a simple function that receives two values and calculates the sum of the squares of each value.

```
CREATE OR REPLACE FUNCTION sum_squares (params IN SDO_RDF_TERM_LIST)
   RETURN SDO_RDF_TERM
   AS
     retTerm    SDO_RDF_TERM;
     sqr1       NUMBER;
     sqr2       NUMBER;
     addVal     NUMBER;
     val1       SDO_RDF_TERM;
     val2       SDO_RDF_TERM;
   BEGIN
     -- Set the return value to null.
     retTerm := SDO_RDF_TERM(NULL,NULL,NULL,NULL,NULL);
     -- Obtain the data from the first two parameters.
     val1 := params(1);
     val2 := params(2);
     -- Convert the value stored in the sdo_rdf_term to number.
     -- If any exception occurs, return the null value.
     BEGIN
       sqr1 := TO_NUMBER(val1.value_name);
       sqr2 := TO_NUMBER(val2.value_name);
       EXCEPTION WHEN OTHERS THEN RETURN retTerm;
     END;
     -- Compute the square sum of both values.
       addVal := (sqr1 * sqr1) + (sqr2 * sqr2);
     -- Set the return value to the desired rdf term type.
     retTerm := SDO_RDF_TERM('LIT',to_char(addVal),
                'http://www.w3.org/2001/XMLSchema#integer','',NULL);
     - Return the new value.
     RETURN retTerm;
END;
/
SHOW ERRORS;
```

Note that the `sum_squares` function in Example 9-1 does not verify the data type of the value received. It is intended as a demonstration only, and relies on TO_NUMBER to obtain the numeric value stored in the VALUE_NAME field of SDO_RDF_TERM.

Example 9-2   User-Defined Function Used in a FILTER Clause

Example 9-2 shows the `sum_squares` function (from Example 9-1) used in a FILTER clause.

```
SELECT s, o
FROM table(sem_match(
```

```
'SELECT  ?s ?o
 WHERE { ?s ?p ?o
 FILTER (<http://xmlns.oracle.com/rdf/extensions/schema.sum_squares>(?o,?o) > 2)}',
sem_models('MYMODEL'),null,null,null,null,'',null,null,'RDFUSER','NET1'));
```

The query in Example 9-2 returns the following result:

```
s                    o
-------------------- --------------------
b                    1.5
c                    3
d                    4
```

Example 9-3 User-Defined Function Used in a SELECT Expression

Example 9-3 shows the `sum_squares` function (from Example 9-1) used in an expression in the SELECT clause.

```
SELECT s, o, sqr_sum
FROM table(sem_match(
'SELECT  ?s ?o
       (<http://xmlns.oracle.com/rdf/extensions/schema.sum_squares>(?o,?o) AS
        ?sqr_sum)
 WHERE { ?s ?p ?o }',
sem_models('MYMODEL'),null,null,null,null,'',null,null,'RDFUSER','NET1'));
```

The query in Example 9-3 returns the following result:

```
s                    o                    sqr_sum
-------------------- -------------------- --------------------
a                    1                    2
b                    1.5                  4.5
c                    3                    18
d                    4                    32
```

Example 9-4 User-Defined Function Used in a BIND Operation

Example 9-4 shows the `sum_squares` function (from Example 9-1) used in a BIND operation.

```
SELECT s, o, sqr_sum
FROM table(sem_match(
'SELECT  ?s ?o ?sqr_sum
 WHERE { ?s ?p ?o .
 BIND (<http://xmlns.oracle.com/rdf/extensions/schema.sum_squares>(?o,?o) AS
       ?sqr_sum)}',
sem_models('MYMODEL'),null,null,null,null,'',null,null,'RDFUSER','NET1'));
```

The query in Example 9-4 returns the following result:

```
s                    o                    sqr_sum
-------------------- -------------------- --------------------
a                    1                    2
b                    1.5                  4.5
c                    3                    18
d                    4                    32
```

9.2.3 API Support for User-Defined Aggregates

User-defined aggregates are implemented by defining a PL/SQL object type that implements a set of interface methods. After the user-defined aggregate is created, a specific URI is used to invoke it.

9.2.3.1 ODCIAggregate Interface

User-defined aggregates use the `ODCIAggregate` PL/SQL interface. For more detailed information about this interface, see the chapter about user-defined aggregate functions in *Oracle Database Data Cartridge Developer's Guide*.

The `ODCIAggregate` interface is implemented by a PL/SQL object type that implements four main functions:

- `ODCIAggregateInitialize`
- `ODCIAggregateIterate`
- `ODCIAggregateMerge`
- `ODCIAggregateTerminate`

As with user-defined functions (described in API Support for User-Defined Functions), user-defined aggregates receive an arbitrary number of RDF term arguments, which are passed in as an SDO_RDF_TERM_LIST object, and return a single RDF term value, which is represented as an SDO_RDF_TERM object.

This scheme results in the following signatures for the PL/SQL `ODCIAggregate` interface functions (with *my_aggregate_obj_type* representing the actual object type name):

```
STATIC FUNCTION ODCIAggregateInitialize(
        sctx IN OUT my_aggregate_obj_type)
RETURN NUMBER

MEMBER FUNCTION ODCIAggregateIterate(
        self       IN OUT my_aggregate_obj_type
       ,value      IN     SDO_RDF_TERM_LIST)
RETURN NUMBER

MEMBER FUNCTION ODCIAggregateMerge(
        self IN OUT my_aggregate_obj_type
       ,ctx2 IN     my_aggregate_obj_type)
RETURN NUMBER

MEMBER FUNCTION ODCIAggregateTerminate (
        self IN my_aggregate_obj_type
       ,return_value OUT SDO_RDF_TERM
       ,flags IN NUMBER)
RETURN NUMBER
```

9.2.3.2 Invoking User-Defined Aggregates

After a user-defined aggregate is implemented in PL/SQL, it can be invoked from a SPARQL query by referring to an aggregate URI constructed from the prefix `<http://xmlns.oracle.com/rdf/aggExtensions/>` followed by *`schema_name.aggregate_name`*. The following is an example aggregate URI:

```
<http://xmlns.oracle.com/rdf/aggExtensions/schema.my_aggregate>(arg_1, …, arg_n)
```

The DISTINCT modifier can be used with user-defined aggregates, as in the following example:

```
<http://xmlns.oracle.com/rdf/aggExtensions/schema.my_aggregate>(DISTINCT arg_1)
```

In this case, only distinct argument values are passed to the aggregate. Note, however, that the DISTINCT modifier can only be used with aggregates that have exactly one argument.

9.2.3.3 User-Defined Aggregate Examples

This section presents examples of implementing and using a user-defined aggregate. For the examples, assume that the following data, presented here in N-triple format, exists inside a model called `MYMODEL`:

```
<a>  <p>  "1.0"^^xsd:double .
<b>  <p>  "1.5"^^xsd:float .
<c>  <p>  "3"^^xsd:decimal .
<c>  <p>  "4"^^xsd:decimal .
<d>  <p>  "4"^^xsd:string .
```

Example 9-5   User-Defined Aggregate Implementation

Example 9-5 shows the implementation of a simple user-defined aggregate (`countSameType`). This aggregate has two arguments: the first is any RDF term, and the second is a constant data type URI. The aggregate counts how many RDF terms from the first argument position have a data type equal to the second argument.

```
-- Aggregate type creation
CREATE OR REPLACE TYPE countSameType authid current_user AS OBJECT(

count NUMBER, -- Variable to store the number of same-type terms.

-- Mandatory Functions for aggregates
STATIC FUNCTION ODCIAggregateInitialize(
        sctx IN OUT countSameType)
RETURN NUMBER,

MEMBER FUNCTION ODCIAggregateIterate(
         self       IN OUT countSameType
       , value      IN     SDO_RDF_TERM_LIST)
RETURN NUMBER,

MEMBER FUNCTION ODCIAggregateMerge(
        self IN OUT countSameType
       ,ctx2 IN     countSameType)
RETURN NUMBER,

MEMBER FUNCTION ODCIAggregateTerminate (
        self IN          countSameType
       ,return_value OUT SDO_RDF_TERM
       ,flags IN NUMBER)
RETURN NUMBER
);
/
SHOW ERRORS;

-- Interface function for the user-defined aggregate
CREATE OR REPLACE FUNCTION countSameAs (input SDO_RDF_TERM_LIST) RETURN SDO_RDF_TERM
PARALLEL_ENABLE AGGREGATE USING countSameType;
/
show errors;

-- User-defined aggregate body
CREATE OR REPLACE TYPE BODY countSameType IS
```

```
STATIC FUNCTION ODCIAggregateInitialize(
         sctx            IN OUT countSameType)
RETURN NUMBER IS
BEGIN
  sctx := countSameType (0); -- Aggregate initialization
  RETURN ODCIConst.Success;
END;

MEMBER FUNCTION ODCIAggregateIterate(
         self           IN OUT countSameType
       , value          IN     SDO_RDF_TERM_LIST )
RETURN NUMBER IS
BEGIN
  -- Increment count if the first argument has a literal type
  -- URI equal to the value of the second argument
  IF (value(1).literal_type = value(2).value_name) THEN
    self.count := self.count + 1;
  END IF;
  RETURN ODCIConst.Success;
END;

MEMBER FUNCTION ODCIAggregateMerge(
         self            IN OUT countSameType
        ,ctx2            IN countSameType)
RETURN NUMBER IS
BEGIN
  -- Sum count to merge parallel threads.
  self.count := self.count + ctx2.count;
  RETURN ODCIConst.Success;
END;

MEMBER FUNCTION ODCIAggregateTerminate(
         self            IN  countSameType
        ,return_value    OUT SDO_RDF_TERM
        ,flags           IN  NUMBER)
RETURN NUMBER IS
BEGIN
   -- Set the return value
   return_value := SDO_RDF_TERM('LIT',to_char(self.count),
     'http://www.w3.org/2001/XMLSchema#decimal',NULL,NULL); RETURN  ODCIConst.Success;
END;

END;
/
SHOW ERRORS;
```

Example 9-6 User-Defined Aggregate Used Without a GROUP BY Clause

Example 9-6 shows the `countSameType` aggregate (from Example 9-5) used over an entire query result group.

```
FROM o
from table(sem_match(
'SELECT
 (<http://xmlns.oracle.com/rdf/aggExtensions/schema.countSameType>(?o,xsd:decimal)
  AS ?o)
 WHERE { ?s ?p ?o }',
sem_models('MYMODEL'),null,null,null,null,'',null,null,'RDFUSER','NET1'));
```

The query in Example 9-6 returns the following result:

```
o
--------------------
2
```

Example 9-7 User-Defined Aggregate Used With a GROUP BY Clause

Example 9-7 shows the `countSameType` aggregate (from Example 9-5) used over a set of groups formed from a GROUP BY clause.

```
select s, o
from table(sem_match(
'SELECT ?s
 (<http://xmlns.oracle.com/rdf/aggExtensions/schema.countSameType>(?o,xsd:decimal)
  AS ?o)
 WHERE { ?s ?p ?o } GROUP BY ?s',
sem_models('MYMODEL'),null,null,null,null,'',null,null,'RDFUSER','NET1'));
```

The query in Example 9-7 returns the following result:

```
s                    o
-------------------- --------------------
a                    0
b                    0
c                    2
d                    0
```

10 RDF Views: Relational Data as RDF

You can create and use RDF views over relational data in RDF Graph.

Relational data is viewed as virtual RDF triples using one of the two forms of RDB2RDF mapping described in W3C documents on Direct Mapping and R2RML mapping:

- *R2RML: RDB to RDF Mapping Language*, W3C Recommendation (`http://www.w3.org/TR/r2rml/`)
- *A Direct Mapping of Relational Data to RDF*, W3C Recommendation (`http://www.w3.org/TR/rdb-direct-mapping/`)

This chapter explains the following topics:

- Why Use RDF Views on Relational Data?
  Using RDF views on relational data enables you to query relational data using SPARQL and integrate data available from different sources.
- API Support for RDF Views
  Subprograms are included in the SEM_APIS package for creating, dropping, and exporting (that is, materializing the content of) RDF views.
- Example: Using an RDF View Graph with Direct Mapping
  This section shows an example of using an RDF view graph with direct mapping.
- Combining Native RDF Data with Virtual RDB2RDF Data
  You can combine native triple data with virtual RDB2RDF triple data (from an RDF view graph) in a single SEM_MATCH query by means of the SERVICE keyword.

10.1 Why Use RDF Views on Relational Data?

Using RDF views on relational data enables you to query relational data using SPARQL and integrate data available from different sources.

You can exploit the advantages of relational data without the need for physical storage of the RDF triples that correspond to the relational data.

The simplest way to create a mapping of relational data to RDF data is by calling the SEM_APIS.CREATE_RDFVIEW_GRAPH procedure to create an RDF view graph, supplying the list of tables or views whose content you would like to be viewed as RDF. This provides a direct mapping of those relational tables or views.

To get a more customized mapping, you can call the SEM_APIS.CREATE_RDFVIEW_GRAPH procedure to create an RDF view graph, supplying the R2RML mapping (using Turtle or N-Triple syntax) with the `r2rml_string` parameter.

10.2 API Support for RDF Views

Subprograms are included in the SEM_APIS package for creating, dropping, and exporting (that is, materializing the content of) RDF views.

An RDF view graph is created as an RDF graph, but the RDF graph physically contains only the mapping metadata. The actual data remains in the relational tables for which the RDF view graph has been created. (The SEM_APIS subprograms are documented in SEM_APIS Package Subprograms.)

Once an RDF view graph is created, you can also materialize the RDF triples into a staging table by using the SEM_APIS.EXPORT_RDFVIEW_GRAPH subprogram.

For the examples throughout this chapter, assume that the relational tables, `EMP` and `DEPT`, are present in the `TESTUSER` schema (see Section 10.3 for the definitions of these two tables). Also, assume that a schema-private network, named NET1 and owned by the RDFUSER schema, already exists and RDFUSER has READ privilege on these two tables.

For the example illustrating the use of exporting of RDF triples, assume that the staging table to which the materialized RDF triples will be stored are owned by `TESTUSER` and the network owner has `INSERT` privilege on that table.

- Creating an RDF View Graph with Direct Mapping
- Creating an RDF View Graph with R2RML Mapping
- Dropping an RDF View Graph
- Exporting Virtual Content of an RDF View Graph into a Staging Table

10.2.1 Creating an RDF View Graph with Direct Mapping

Example 10-1 creates an RDF view graph using direct mapping of two tables, EMP and DEPT (see Section 10.3 for the definitions of these two tables), with a base prefix of `http://empdb/` in a schema-private network. The (virtual) RDF terms are generated according to A Direct Mapping of Relational Data to RDF, W3C Recommendation.

Example 10-1 Creating an RDF View Graph with Direct Mapping in a Schema-Private Network

```
BEGIN
  sem_apis.create_rdfview_graph(
    rdf_graph_name => 'empdb_model',
    tables => SYS.ODCIVarchar2List('"TESTUSER"."EMP"', '"TESTUSER"."DEPT"'),
    prefix => 'http://empdb/',
    options => 'KEY_BASED_REF_PROPERTY=T',
    network_owner=>'RDFUSER',
    network_name=>'NET1'
  );
END;
/
```

To see the properties that are generated, enter the following query:

```
SELECT p
  FROM TABLE(SEM_MATCH(
    'SELECT DISTINCT ?p {?s ?p ?o} ORDER BY ?p',
    SEM_Models('empdb_model'),
    NULL, NULL, NULL, NULL,
    NULL, NULL, NULL,
   'RDFUSER', 'NET1'));
P
------------------------------------------------------------------------------
--
http://empdb/TESTUSER.EMP#EMPNO
```

```
http://empdb/TESTUSER.EMP#JOB
http://empdb/TESTUSER.EMP#ENAME
http://empdb/TESTUSER.EMP#DEPTNO
http://www.w3.org/1999/02/22-rdf-syntax-ns#type
http://empdb/TESTUSER.EMP#ref-DEPTNO
http://empdb/TESTUSER.DEPT#DEPTNO
http://empdb/TESTUSER.DEPT#DNAME
http://empdb/TESTUSER.DEPT#LOC

9 rows selected.
```

10.2.2 Creating an RDF View Graph with R2RML Mapping

You can create an RDF view graph using the two tables EMP and DEPT, but with your own customizations, by creating an R2RML mapping document specified using Turtle, as shown:

```
@prefix rr: <http://www.w3.org/ns/r2rml#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix ex: <http://example.com/ns#>.

ex:TriplesMap_Dept
    rr:logicalTable [ rr:tableName "TESTUSER.DEPT" ];
    rr:subjectMap [
        rr:template "http://data.example.com/department/{DEPTNO}";
        rr:class ex:Department;
    ];
    rr:predicateObjectMap [
        rr:predicate ex:deptNum;
        rr:objectMap [ rr:column "DEPTNO" ; rr:datatype xsd:integer ];
    ];
    rr:predicateObjectMap [
        rr:predicate ex:deptName;
        rr:objectMap [ rr:column "DNAME" ];
    ];
    rr:predicateObjectMap [
        rr:predicate ex:deptLocation;
        rr:objectMap [ rr:column "LOC" ];
    ].

ex:TriplesMap_Emp
    rr:logicalTable [ rr:tableName "TESTUSER.EMP" ];
    rr:subjectMap [
        rr:template "http://data.example.com/employee/{EMPNO}";
        rr:class ex:Employee;
    ];
    rr:predicateObjectMap [
        rr:predicate ex:empNum;
        rr:objectMap [ rr:column "EMPNO" ; rr:datatype xsd:integer ];
    ];
    rr:predicateObjectMap [
        rr:predicate ex:empName;
        rr:objectMap [ rr:column "ENAME" ];
    ];
    rr:predicateObjectMap [
        rr:predicate ex:jobType;
```

```
        rr:objectMap [ rr:column "JOB" ];
    ];
    rr:predicateObjectMap [
        rr:predicate ex:worksForDeptNum;
        rr:objectMap [ rr:column "DEPTNO" ; rr:dataType xsd:integer ];
    ];
    rr:predicateObjectMap [
        rr:predicate ex:worksForDept;
        rr:objectMap [
          rr:parentTriplesMap ex:TriplesMap_Dept ;
          rr:joinCondition [ rr:child "DEPTNO"; rr:parent "DEPTNO" ]]].
```

Example 10-2 Creating an RDF View Graph with an R2RML Mapping String

The following example creates an RDF view graph directly from an R2RML string, using the preceding R2RML mapping:

```
DECLARE
  r2rmlStr CLOB;

BEGIN

  r2rmlStr :=
   '@prefix rr: <http://www.w3.org/ns/r2rml#>. '||
   '@prefix xsd: <http://www.w3.org/2001/XMLSchema#>. '||
   '@prefix ex: <http://example.com/ns#>. '||'

    ex:TriplesMap_Dept
        rr:logicalTable [ rr:tableName "TESTUSER.DEPT" ];
        rr:subjectMap [
            rr:template "http://data.example.com/department/{DEPTNO}";
            rr:class ex:Department;
        ];
        rr:predicateObjectMap [
            rr:predicate ex:deptNum;
            rr:objectMap [ rr:column "DEPTNO" ; rr:datatype xsd:integer ];
        ];
        rr:predicateObjectMap [
            rr:predicate ex:deptName;
            rr:objectMap [ rr:column "DNAME" ];
        ];
        rr:predicateObjectMap [
            rr:predicate ex:deptLocation;
            rr:objectMap [ rr:column "LOC" ];
        ].'||'

    ex:TriplesMap_Emp
        rr:logicalTable [ rr:tableName "TESTUSER.EMP" ];
        rr:subjectMap [
            rr:template "http://data.example.com/employee/{EMPNO}";
            rr:class ex:Employee;
        ];
        rr:predicateObjectMap [
            rr:predicate ex:empNum;
            rr:objectMap [ rr:column "EMPNO" ; rr:datatype xsd:integer ];
        ];
```

```
        rr:predicateObjectMap [
            rr:predicate ex:empName;
            rr:objectMap [ rr:column "ENAME" ];
        ];
        rr:predicateObjectMap [
            rr:predicate ex:jobType;
            rr:objectMap [ rr:column "JOB" ];
        ];
        rr:predicateObjectMap [
            rr:predicate ex:worksForDeptNum;
            rr:objectMap [ rr:column "DEPTNO" ; rr:dataType xsd:integer ];
        ];
        rr:predicateObjectMap [
            rr:predicate ex:worksForDept;
            rr:objectMap [
              rr:parentTriplesMap ex:TriplesMap_Dept ;
              rr:joinCondition [ rr:child "DEPTNO"; rr:parent "DEPTNO" ]]].';

  sem_apis.create_rdfview_graph(
    rdf_graph_name => 'empdb_model',
    tables => NULL,
    r2rml_string => r2rmlStr,
    r2rml_string_fmt => 'TURTLE',
    network_owner=>'RDFUSER',
    network_name=>'NET1'
  );

END;
/
```

10.2.3 Dropping an RDF View Graph

An RDF view graph can be dropped using the SEM_APIS.DROP_RDFVIEW_GRAPH procedure, as shown in Example 10-3.

Example 10-3 Dropping an RDF View graph

```
BEGIN
  sem_apis.drop_rdfview_model(
    rdf_graph_name => 'empdb_model',
    network_owner =>'RDFUSER',
    network_name =>'NET1'
  );
END;
/
```

10.2.4 Exporting Virtual Content of an RDF View Graph into a Staging Table

The content of an RDF view graph is virtual; that is, the RDF triples corresponding to the underlying relational data, as mapped by direct mapping or R2RML mapping, are not materialized and stored anywhere. The SEM_APIS.EXPORT_RDFVIEW_GRAPH subprogram lets you materialize the virtual RDF triples of an RDF view graph into a staging table. The staging table can then be used for loading into an RDF graph.

Example 10-4 Exporting an RDF View Graph in a Schema-Private Network

Example 10-4 materializes (in N-Triples format) the content of RDF view `empdb_model` into the staging table `TESTUSER.R2RTAB`.

```
BEGIN
  sem_apis.export_rdfview_graph(
    rdf_graph_name => 'empdb_model',
    rdf_table_owner => 'TESTUSER',
    rdf_table_name => 'R2RTAB',
    network_owner => 'RDFUSER',
    network_name => 'NET1'
  );
END;
PL/SQL procedure successfully completed.
```

10.3 Example: Using an RDF View Graph with Direct Mapping

This section shows an example of using an RDF view graph with direct mapping.

Perform the following steps for creating and using an RDF view graph with direct mapping.

1. Create two relational tables, `EMP` and `DEPT`, in the `TESTUSER` schema and grant `READ` privilege on these two tables to `RDFUSER`.

```
-- Use the following relational tables.
CREATE TABLE TESTUSER.dept (
  deptno NUMBER CONSTRAINT pk_DeptTab_deptno PRIMARY KEY,
  dname VARCHAR2(30),
  loc VARCHAR2(30)
);

CREATE TABLE TESTUSER.emp (
  empno NUMBER PRIMARY KEY,
  ename VARCHAR2(30),
  job VARCHAR2(20),
  deptno NUMBER REFERENCES dept (deptno)
);

GRANT READ ON TESTUSER.dept TO RDFUSER;

GRANT READ ON TESTUSER.emp TO RDFUSER;
```

2. Insert data into the tables.

```
-- Insert some data.

INSERT INTO TESTUSER.dept (deptno, dname, loc)
  VALUES (1, 'Sales', 'Boston');
INSERT INTO TESTUSER.dept (deptno, dname, loc)
  VALUES (2, 'Manufacturing', 'Chicago');
INSERT INTO TESTUSER.dept (deptno, dname, loc)
  VALUES (3, 'Marketing', 'Boston');

INSERT INTO TESTUSER.emp (empno, ename, job, deptno)
```

```
  VALUES (1, 'Alvarez', 'SalesRep', 1);
INSERT INTO TESTUSER.emp (empno, ename, job, deptno)
  VALUES (2, 'Baxter', 'Supervisor', 2);
INSERT INTO TESTUSER.emp (empno, ename, job, deptno)
  VALUES (3, 'Chen', 'Writer', 3);
INSERT INTO TESTUSER.emp (empno, ename, job, deptno)
  VALUES (4, 'Davis', 'Technician', 2);
```

3. Connect as `RDFUSER` and create an RDF view graph, `empdb_model`, using direct mapping of the two tables created and populated in the preceding steps.

```
-- Create an RDF view graph using direct mapping of two tables, EMP and
DEPT,
-- with a base prefix of http://empdb/.
-- Specify KEY_BASED_REF_PROPERTY=T for the options parameter.


BEGIN
  sem_apis.create_rdfview_graph(
    rdf_graph_name => 'empdb_model',
    tables => SYS.ODCIVarchar2List('"TESTUSER"."EMP"',
'"TESTUSER"."DEPT"'),
    prefix => 'http://empdb/',
    options => 'KEY_BASED_REF_PROPERTY=T'
    network_owner=>'RDFUSER',
    network_name=>'NET1'
  );
END;
/
```

4. Query the newly created RDF view graph using a SEM_MATCH-based SQL query.

```
SELECT emp
  FROM TABLE(SEM_MATCH(
    'PREFIX dept: <http://empdb/TESTUSER.DEPT#>
     PREFIX emp: <http://empdb/TESTUSER.EMP#>
     SELECT ?emp {?emp emp:ref-DEPTNO ?dept . ?dept dept:LOC "Boston"}',
    SEM_Models('empdb_model'),
    NULL,
    NULL,
    NULL, NULL,NULL, NULL,NULL, 'RDFUSER', 'NET1'));

EMP
---------------------------------------------------------------------------
-----
http://empdb/TESTUSER.EMP/EMPNO=1
http://empdb/TESTUSER.EMP/EMPNO=3
```

The query shown in this step is functionally comparable to:

```
SQL> SELECT e.empno FROM emp e, dept d WHERE e.deptno = d.deptno AND d.loc
= 'Boston';

     EMPNO
----------
```

```
1
3
```

10.4 Combining Native RDF Data with Virtual RDB2RDF Data

You can combine native triple data with virtual RDB2RDF triple data (from an RDF view graph) in a single SEM_MATCH query by means of the SERVICE keyword.

The SERVICE keyword (explained in Graph Patterns: Support for SPARQL 1.1 Federated Query) is overloaded through the use of special SERVICE URLs that signify local (virtual) RDF data. The following prefixes are used to denote special SERVICE URLs:

- Native RDF graphs - `oram`: <http://xmlns.oracle.com/models/>
- Native RDF graph collections - `oravm`: <http://xmlns.oracle.com/virtual_models/>
- RDB2RDF models - `orardbm`: <http://xmlns.oracle.com/rdb_models/>

Example 10-5 Querying Multiple Data Sets

Example 10-5 queries multiple data sets. In this query, the first triple pattern `{ ?x rdf:type :Person }` will go against an RDF graph `m1` as usual, but `{ ?x :name ?name }` will go against the local RDF graph `m2`, and { `?x emp:JOB ?job }` will go against the local RDB2RDF model `empdb_model`.

```
SELECT * FROM TABLE (SEM_MATCH(
'PREFIX    : <http://people.org/>
 PREFIX emp: <http://empdb/TESTUSER.EMP#>
 SELECT ?x ?name ?job
 WHERE {
   ?x rdf:type :Person .
   OPTIONAL { SERVICE oram:m2 { ?x :name ?name } }
   OPTIONAL { SERVICE orardbm:empdb_model { ?x emp:JOB ?job } }
 }',
 SEM_MODELS('m1'), NULL, NULL, NULL, NULL, ' ', NULL, NULL, 'RDFUSER',
'NET1'));
```

Overloaded SERVICE use is only allowed with a single RDF graph specified in the `models` argument of SEM_MATCH. Overloaded SERVICE queries do not allow multiple RDF graphs or a rulebase as input. An RDF graph collection that contains multiple RDF graphs and/or inferred graphs should be used instead for such combinations. In addition, the `index_status` argument for SEM_MATCH will only check the inferred graph contained in the RDF graph collection passed as input in the `models` parameter. This means the status of inferred graphs that are referenced in overloaded SERVICE calls will not be checked.

Example 10-6 queries two data sets: the `empdb_model` from Example: Using an RDF View Graph with Direct Mapping and a native RDF graph named `people`.

Example 10-6 Querying Virtual RDB2RDF Data and Native RDF Data in a Schema-Private Network

```
-- Create native model people --
 EXECUTE SEM_APIS.CREATE_RDF_GRAPH('people', NULL, NULL,
network_owner=>'rdfuser', network_name=>'net1');

BEGIN
```

```
  SEM_APIS.UPDATE_RDF_GRAPH('people',
   'PREFIX peop: <http://people.org/>
    INSERT DATA {
       <http://empdb/TESTUSER.EMP/EMPNO=1> peop:age 35 .
       <http://empdb/TESTUSER.EMP/EMPNO=2> peop:age 39 .
       <http://empdb/TESTUSER.EMP/EMPNO=3> peop:age 30 .
       <http://empdb/TESTUSER.EMP/EMPNO=4> peop:age 42 .
    } ');
END;
/
COMMIT;

-- Querying multiple datasets --
SELECT emp, age
  FROM TABLE(SEM_MATCH(
    'PREFIX dept: <http://empdb/TESTUSER.DEPT#>
     PREFIX emp: <http://empdb/TESTUSER.EMP#>
     PREFIX peop: <http://people.org/>
     SELECT ?emp ?age WHERE {
       ?emp peop:age ?age
       SERVICE orardbm:empdb_model { ?emp emp:ref-DEPTNO ?dept . ?dept
dept:LOC "Boston" }
    }',
    SEM_Models('people'),
    NULL,
    NULL,
    NULL, NULL, NULL, NULL, NULL, 'RDFUSER', 'NET1'));
```

The query produces the following output:

```
EMP                                                   AGE
--------------------------------------------------
--------------------------------------------------
http://empdb/TESTUSER.EMP/EMPNO=1                      35
http://empdb/TESTUSER.EMP/EMPNO=3                      30
```

- Nested Loop Pushdown with Overloaded Service

10.4.1 Nested Loop Pushdown with Overloaded Service

Using a nested loop service can improve performance is some scenarios. Consider the following example queries against multiple data sets for a schema-private network. The query finds the properties of all the departments with people who are 35 years old.

```
-- Query example for a schema-private network.

SELECT emp, dept, p, o
  FROM TABLE(SEM_MATCH(
    'PREFIX dept: <http://empdb/TESTUSER.DEPT#>
     PREFIX emp: <http://empdb/TESTUSER.EMP#>
     PREFIX peop: <http://people.org/>
     SELECT * WHERE{
       ?emp peop:age 35
       SERVICE orardbm:empdb_model{ ?emp emp:ref-DEPTNO ?dept . ?dept ?p ?o }
```

```
      }',
      SEM_Models('people'),
      NULL,
      NULL,
      NULL, NULL, NULL, NULL, NULL, 'RDFUSER', 'NET1'));
```

The preceding query produces the following output:

```
EMP                                DEPT
P                                                 O
--------------------------------- -----------------------------------
------------------------------------------------  --------------------------
http://empdb/TESTUSER.EMP/EMPNO=1   http://empdb/TESTUSER.DEPT/DEPTNO=1
http://empdb/TESTUSER.DEPT#DEPTNO               1
http://empdb/TESTUSER.EMP/EMPNO=1   http://empdb/TESTUSER.DEPT/DEPTNO=1
http://empdb/TESTUSER.DEPT#DNAME                Sales
http://empdb/TESTUSER.EMP/EMPNO=1   http://empdb/TESTUSER.DEPT/DEPTNO=1
http://empdb/TESTUSER.DEPT#LOC                  Boston
http://empdb/TESTUSER.EMP/EMPNO=1   http://empdb/TESTUSER.DEPT/DEPTNO=1
http://www.w3.org/1999/02/22-rdf-syntax-ns#type  http://empdb/TESTUSER.DEPT
```

To get all the results that match for given graph pattern, first the triple pattern `{ ?emp peop:age 35 }` is matched against the RDF graph `people`, then the triple patterns `{ ?emp emp:ref-DEPTNO ?d . ?d dept:DNAME ?dept }` are matched against the RDF graph `empdb_model`, and finally the results are joined. Assume that there is only one 35-year-old person in the RDF graph `people`, but there are 100,000 triples with information about departments. Obviously, a strategy that retrieves all the results is not the most efficient, and query may have poor performance because a large number of results that need to be processed before being joined with the rest of the query.

An nested-loop service can improve performance in this case. If the hint `OVERLOADED_NL=T` is used, the results of the first part of the query are computed and the SERVICE pattern is executed procedurally in a nested loop once for each `?emp` value from the root triple pattern. The `?emp` subject variable in the SERVICE pattern is replaced with a constant from the root triple pattern in each execution. This effectively pushes the join condition down into the SERVICE clause.

The following example shows the use of the `OVERLOADED_NL=T` hint for the preceding query.

```
SELECT emp, dept, p, o
  FROM TABLE(SEM_MATCH(
    'PREFIX dept: <http://empdb/TESTUSER.DEPT#>
     PREFIX emp: <http://empdb/TESTUSER.EMP#>
     PREFIX peop: <http://people.org/>
     SELECT * WHERE{
       ?emp peop:age 35
       SERVICE orardbm:empdb_model { ?emp emp:ref-DEPTNO ?dept . ?dept ?p ?o }
     }',
     SEM_Models('people'),
     NULL,
     NULL,
     NULL, NULL,' OVERLOADED_NL=T ', NULL, NULL, 'RDFUSER', 'NET1'));
```

The hint `OVERLOADED_NL=T` can be specified among SEM_MATCH options or among inline comments for a given SERVICE graph.

11

Property Graph Views on RDF Graphs

Oracle Graph supports the property graph data model in addition to the RDF graph data model.

The property graph data model is simpler than the RDF data model in that it has no concept of global resource identification (that is, no URIs) or formal semantics and inference. In addition, property graphs allow direct association of properties (key-value pairs) with edges. RDF, by contrast, needs reification or a quad data model to associate properties with edges (RDF triples).

The property graph feature (see Introduction to Property Graphs in *Oracle Database Graph Developer's Guide for Property Graph*) of Oracle Database supports analytics capabilities with nearly 60 pre-built algorithms. You can avail this feature with RDF graphs by creating property graph views on the RDF graphs.

The `CREATE PROPERTY GRAPH` DDL statement supported by the Property Graph Query Language (PGQL) (see Creating a Property Graph Using PGQL) creates a property graph using relational data from the database. The vertices and the edges of the property graph are derived from the vertex and edge tables provided in the DDL statement. So you can run SEM_MATCH queries on your RDF data to create the database views that represent the the vertex and edge tables.

Also, note the following:

- The vertex and edge views need a primary key column and attributes. If your SPARQL pattern uses a multi-valued property, then you may have repeated rows with the same primary key (usually a repeated subject in a vertex table). For such properties, you need to make them edges or use some aggregate like `JSON_ARRAYAGG` to collapse the multi-valued property into a single row.
- The graph server (PGX) which runs the graph algorithms cannot handle composite primary keys. Therefore, you need to build a single key column for edge tables instead of simply using (`sourceId`, `destinationId`) as key.

Example 11-1 Create a PGQL Property Graph from RDF Data

Prerequistes: The following example uses the Moviestream RDF data and assumes that this data is loaded into an RDF graph called MOVIESTREAM in a network named RDF_NETWORK owned by RDFUSER. See Bulk Loading RDF Data Using SQL Developer for using SQL Developer to bulk load RDF data.

Perform the following steps to create a PGQL property graph using RDF data:

1. Run SEM_MATCH queries to create views to represent the vertex and edge tables.
   The following example code generates the database views corresponding to the vertex and edge tables as shown:
   - **Vertex Tables:** `MOVIE`, `GENRE`
   - **Edge Table:** `HAS_GENRE`

   ```
   /* Vertex Table: Movie
   http://www.example.com/moviestream/Movie
     - http://www.example.com/moviestream/title
   ```

```
  - http://www.example.com/moviestream/sku
  - http://www.example.com/moviestream/year
  - http://www.example.com/moviestream/views
  - http://www.example.com/moviestream/summary
  - http://www.example.com/moviestream/runtimeInMin
  - http://www.example.com/moviestream/grossInUSD
  - http://www.example.com/moviestream/budgetInUSD
  - http://www.example.com/moviestream/openingDate
*/

create or replace view movie(id, title, summary, year, openingDate,
runtimeinMin, grossInUSD, budgetInUSD, views) as
select movie$rdfvid id,
       title,
       summary,
       cast(year as number default null on conversion error) year,
       to_timestamp(openingDate default null on conversion error, 'SYYYY-
MM-DD') openingDate,
       cast(runtimeInMin as number default null on conversion error)
runtimeinMin,
       cast(grossInUSD as number default null on conversion error)
grossInUSD,
       cast(budgetInUSD as number default null on conversion error)
budgetInUSD,
       cast(views as number default null on conversion error) views
from table(sem_match(
'PREFIX ms: <http://www.example.com/moviestream/>
 SELECT *
 WHERE {
   ?movie ms:title ?title .
   OPTIONAL { ?movie ms:summary ?summary }
   OPTIONAL { ?movie ms:sku ?sku }
   OPTIONAL { ?movie ms:year ?year }
   OPTIONAL { ?movie ms:openingDate ?openingDate }
   OPTIONAL { ?movie ms:runtimeInMin ?runtimeInMin }
   OPTIONAL { ?movie ms:grossInUSD ?grossInUSD }
   OPTIONAL { ?movie ms:budgetInUSD ?budgetInUSD }
   OPTIONAL { ?movie ms:views ?views }
 }',
sem_models('moviestream'),
null,null,null,null,
' DO_UNESCAPE=T ',
null,null,
'RDFUSER','RDF_NETWORK'));

/* Vertex Table: Genre

http://www.example.com/moviestream/Genre
  - http://www.example.com/moviestream/genreName
*/
create or replace view genre(id, genreName) as
select genre$rdfvid id, genreName
from table(sem_match(
'PREFIX ms: <http://www.example.com/moviestream/>
 SELECT ?genre ?genreName
 WHERE {
```

```
    ?genre ms:genreName ?genreName . }',
sem_models('moviestream'),
null,null,null,null,
' DO_UNESCAPE=T ',
null,null,
'RDFUSER','RDF_NETWORK'));

/*
Edge Table: has_genre
(:Movie) -[http://www.example.com/moviestream/genre]-> (:Genre)
*/
create or replace view has_genre(id, movieId, genreId) as
select (to_char(movie$rdfvid)||to_char(genre$rdfvid)) as id, movie$rdfvid
movieId, genre$rdfvid genreId
from table(sem_match(
'PREFIX ms: <http://www.example.com/moviestream/>
 SELECT *
 WHERE {
   ?movie ms:genre ?genre .
 }',
sem_models('moviestream'),
null,null,null,null,
' DO_UNESCAPE=T ',
null,null,
'RDFUSER','RDF_NETWORK'));
```

2. Create a PGQL property graph using the views.
   You can create PGQL property graphs using either the Graph Clients that are shipped with the Graph server and Client Release or using SQL Client Tools (SQLcl or SQL Developer).

   The following example creates the `MOVIES` property graph using the **PGQL Worksheet** in SQL Developer.

```
CREATE PROPERTY GRAPH MOVIES
VERTEX TABLES (
  MOVIE KEY(ID) LABEL MOVIE PROPERTIES ARE ALL COLUMNS,
  GENRE KEY(ID) LABEL GENRE PROPERTIES ARE ALL COLUMNS
)
EDGE TABLES (
  HAS_GENRE KEY(ID)
    SOURCE KEY (MOVIEID) REFERENCES MOVIE(ID)
    DESTINATION KEY (GENREID) REFERENCES GENRE(ID)
    LABEL HAS_GENRE PROPERTIES ARE ALL COLUMNS
 ) OPTIONS (PG_PGQL)
```

You can now query, visualize, and run graph algorithms on the property graph.

Using the Graph Server (PGX) to Run Graph Algorithms on RDF Graph and RDF Data Visualization

The graph server (PGX) of Oracle Graph allows you to run graph algorithms on property graphs. Hence, you can load the property graph, which is created using the RDF data in the views (as explained in Example 11-1), into the graph server (PGX) and run graph analytics. In addition, you can also visualize the RDF data using the Graph Visualization web client. Note that you must install the graph server (PGX) for performing these operations.

> **See Also:**
>
> - Oracle Graph Server Installation for installing the graph server (PGX)
> - Oracle Graph Clients
> - Graph Visualization Web Client for running the graph visualization client
> - Executing Built-in Algorithms for the supported built-in algorithms

Example 11-2 Running Graph Algorithms on RDF Graphs and RDF Data Visualization

Prerequisites: Ensure that you meet the following prerequisites for running this example:

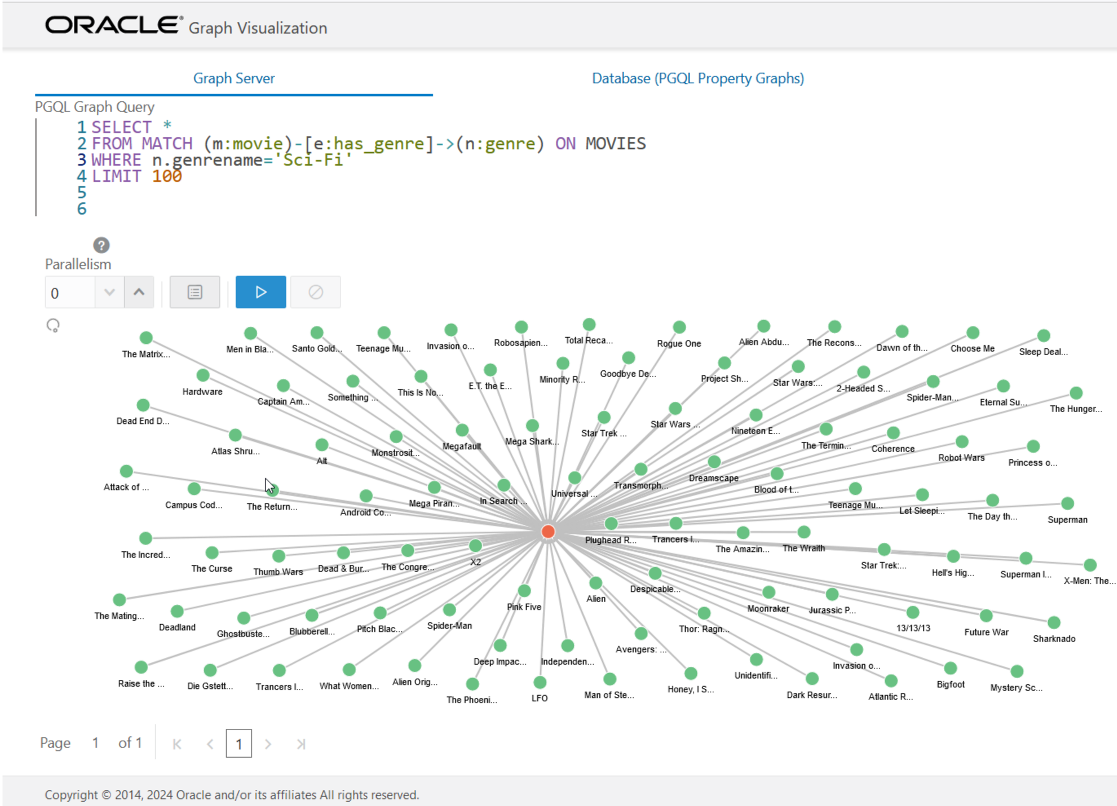
1. Create a PGQL property graph by creating database views on RDF data (see Example 11-1).
2. As a SYSDBA user grant the `GRAPH_DEVELOPER` role to RDFUSER.

The following example uses the Java client to load the property graph into the graph server (PGX) and then runs the PageRank algorithm to list the top 10 movies.

```
For an introduction type: /help intro
Oracle Graph Server Shell 24.2.0
Variables instance, session, and analyst ready to use.
opg4j> var graph = session.readGraphByName("MOVIES",
GraphSource.PG_PGQL,ReadGraphOption.onMissingVertex(OnMissingVertex.IGNORE_EDG
E_LOG_ONCE))
graph ==> PgxGraph[name=MOVIES,N=3823,E=7617,created=1714231298337]
opg4j> analyst.pagerank(graph)
$3 ==> VertexProperty[name=pagerank,type=double,graph=MOVIES]
opg4j> session.queryPgql("SELECT a.title, a.pagerank FROM MATCH (a:movie) ON
MOVIES ORDER BY a.pagerank DESC LIMIT 10").print()
+---------------------------------------------------------+
| title                          | pagerank               |
+---------------------------------------------------------+
| Gang Cops                      | 3.9236201935652636E-5  |
| Blood Street                   | 3.9236201935652636E-5  |
| The Girl on the Train          | 3.9236201935652636E-5  |
| The Girl with the Hungry Eyes  | 3.9236201935652636E-5  |
| Debonair Dancers               | 3.9236201935652636E-5  |
| Honky                          | 3.9236201935652636E-5  |
| Edith's Shopping Bag           | 3.9236201935652636E-5  |
| Believe                        | 3.9236201935652636E-5  |
| Taking Liberties               | 3.9236201935652636E-5  |
| Batman Fights Dracula          | 3.9236201935652636E-5  |
+---------------------------------------------------------+
$5 ==> PgqlResultSetImpl[graph=MOVIES,numResults=10]
```

In addition, you can publish the graph (`opg4j> graph.publish()`) in the graph server (PGX) and run PGQL queries Using the Graph Visualization Application as shown in the following figure. The example visualization shows all the movies that belong to *`Sci-Fi`* genre.

Figure 11-1 RDF Data Visualization



Part II

RDF Graph Server and Query UI

Part II provides information about using RDF Graph Server and Query UI.

This part contains the following chapters:

- Introduction to RDF Graph Server and Query UI
  The RDF Graph Server and Query UI allows you to run SPARQL queries and perform advanced RDF graph data management operations using a REST API and an Oracle JET based query UI.
- RDF Graph Server and Query UI Concepts
  Learn the key concepts for using the RDF Graph Server and Query UI.
- Oracle RDF Graph Query UI
  The Oracle RDF Graph Query UI is an Oracle JET based client that can be used to manage RDF objects from different data sources, and to perform SPARQL queries and updates.

12 Introduction to RDF Graph Server and Query UI

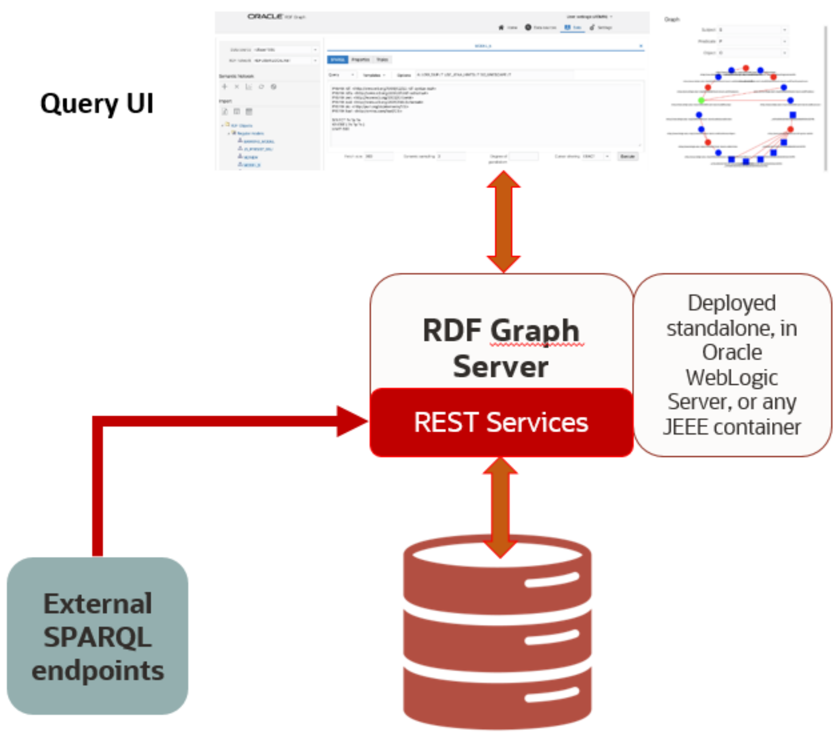
The RDF Graph Server and Query UI allows you to run SPARQL queries and perform advanced RDF graph data management operations using a REST API and an Oracle JET based query UI.

The RDF Graph Server and Query UI consists of RDF RESTful services and a Java EE client application called RDF Graph Query UI. This client serves as an administrative console for Oracle RDF and can be deployed to a Java EE container.

The RDF Graph Server and RDF RESTful services can be used to create a SPARQL endpoint for RDF graphs in Oracle Database.

The following figure shows the RDF Graph Server and Query UI architecture:

Figure 12-1 RDF Graph Server and Query UI



The salient features of the RDF Graph Query UI are as follows:

- Uses RDF RESTful services to communicate with RDF data stores, which can be an Oracle RDF data source or an external RDF data source.

- Allows you to perform CRUD operations on various RDF objects such as private networks, models, rule bases, entailments, network indexes and data types for Oracle data sources.
- Allows you to execute SPARQL queries and update RDF data.
- Provides a graph view of SPARQL query results.
- Uses Oracle JET for user application web pages.

13 RDF Graph Server and Query UI Concepts

Learn the key concepts for using the RDF Graph Server and Query UI.

- Data Sources
  Data sources are repositories of RDF objects.
- RDF Datasets
  Each RDF data source contains metadata information that describe the avaliable RDF objects.
- REST Services
  An RDF REST API allows communication between client and backend RDF data stores.

13.1 Data Sources

Data sources are repositories of RDF objects.

A data source can refer to an Oracle database, or to an external RDF service that can be accessed by an endpoint URL such as Dbpedia or Jena Apache Fuseki. The data source can be defined by generic and as well as specific parameters. Some of the generic parameters are name, type, and description. Specific parameters are JDBC properties (for database data sources) and endpoint base URL (for external data sources).

- Oracle Data Sources
- Endpoint URL Data Sources

13.1.1 Oracle Data Sources

Oracle data sources are defined using JDBC connections. Three types of Oracle JDBC data sources can be defined:

- A JDBC URL data source with standard Oracle JDCB parameters, which include SID or service name, host, port, and user credentials.
- A container JDBC data source that can be defined inside the application Server (WebLogic, Tomcat, or others).
- An Oracle wallet data source that contains the files needed to make the database connection.

The parameters that define an Oracle database data source include:

- **name:** A generic name of the data source.
- **type:** The data source type. For databases, it must be ‘DATABASE’.
- **description (optional):** A generic description of the data source.
- **properties:** Specific mapping parameters with values for data source properties:
  – For a JDBC URL:
    * Database **SID** or **service name**

        * **Host** machine
        * Database listening **port**
        * **User name** and **password** credentials
    – For a container data source:
        * **JNDI name** - Java naming and directory interface (JNDI) name
    – For a wallet data source:
        * A string describing the **wallet service**
        * **User name** and **password** credentials (required if the user credentials are not stored in the wallet)
        * Optional **proxy** details

    For a cloud wallet it is usually an **alias** name stored in the **tnsnames.ora** file, but for a simple wallet it contains the **host**, **port**, and **service** name information.

The following example shows the JSON representation of a JDBC URL data source.

```
{
    "name" : "rdfuser_jdbc_sid",
    "type" : "DATABASE",
    "description" : "",
    "properties" : {
    "host" : "127.0.0.1"
    "sid" : "orcl193"
    "port" : "1524",
    "user" : "rdfuser",
        "password" : "<password>"
    }
}
```

The following example shows the JSON representation of a container data source:

```
{
   "name": "rdfuser_ds_ct",
   "type": "DATABASE",
   "description": "Database Container connection",
   "properties": {
       "jndiName": "jdbc/RDFUSER193c"
   }
}
```

The following example shows the JSON representation of a wallet data source where the credentials are stored in the wallet:

```
{
   "name": "rdfuser_ds_wallet",
   "type": "DATABASE",
   "description": "Database wallet connection",
   "properties": {
       "walletService": "db202002041627_medium"
```

```
    }
}
```

13.1.2 Endpoint URL Data Sources

External RDF data sources are defined using an endpoint URL. In general, each RDF store has a generic URL that accepts SPARQL queries and SPARQL updates. Depending on the RDF store service, it may also provide some capabilities request to retrieve available datasets.

Table 13-1 External Data source Parameters

Parameters	Description
name	A generic name of the data source.
type	The type of the data source. For external data sources, the type must be 'ENDPOINT'.
description	A generic description of the data source.
properties	Specific mapping parameters with values for data source properties: • base URL: the base URL to issue SPARQL queries to RDF store. This is the default URL. • query URL (optional): the URL to execute SPARQL queries. If defined, it will overwrite the base URL value. • update URL (optional): the URL to execute SPARQL updates. If defined, it will overwrite the base URL value. • capabilities (optional): Some RDF stores (like Apache Jena Fuseki) may provide a capabilities URL that returns the datasets available in service. A JSON response is expected in this case. • get URL: the get capabilities URL. • datasets parameter: defines the JSON parameter that contains the RDF datasets information. • dataset parameter name: defines the JSON parameter that contains the RDF dataset name.

The following example shows the JSON representation of a Dbpedia external data source :

```
{
  "name": "dbpedia",
  "type": "ENDPOINT",
  "description": "Dbpedia RDF data - Dbpedia.org",
  "properties": {
      "baseUrl": "http://dbpedia.org/sparql",
      "provider": "Dbpedia"
   }
}
```

The following example shows the JSON representation of a Apache Jena Fuseki external data source. The `${DATASET}` is a parameter that is replaced at run time with the Fuseki dataset name:

```
{
    "name": "Fuseki",
    "type": "ENDPOINT",
    "description": "Jena Fuseki server",
    "properties": {
      "queryUrl": "http://localhost:8080/fuseki/${DATASET}/query",
      "baseUrl": "http://localhost:8080/fuseki",
```

```
      "capabilities": {
        "getUrl": "http://localhost:8080/fuseki/$/server",
        "datasetsParam": "datasets",
        "datasetNameParam": "ds.name"
      },
      "provider": "Apache",
      "updateUrl": "http://localhost:8080/fuseki/${DATASET}/update"
    }
}
```

13.2 RDF Datasets

Each RDF data source contains metadata information that describe the avaliable RDF objects.

The following describes the metadata information defined by each provider.

- **Oracle RDF data sources:** The RDF metadata includes information about the following RDF objects: private networks, models (real, virtual, view), rulebases, entailments, network indexes and datatypes.
- **External RDF providers:** For Apache Jena Fuseki, the metadata includes dataset names. Other external providers may not have a metadata concept, in which case the base URL points to generic (default) metadata.

RDF datasets point to one or more RDF objects available in the RDF data source. A dataset definition is used in SPARQL query requests. Each provider has its own set of properties to describe the RDF dataset.

The following are a few examples of a JSON representation of a dataset.

Oracle RDF dataset definition:

```
[
   {
     "networkOwner": "RDFUSER",
     "networkName": "MYNET",
     "models": ["M1"]
   }
]
```

Apache RDF Jena Fuseki dataset definition:

```
[
   {
     "name": "dataset1"
   }
]
```

For RDF stores that do not have a specific dataset, a simple JSON {} or a 'Default' name as shown for Apache Jena Fuseki in the above example can be used.

13.3 REST Services

An RDF REST API allows communication between client and backend RDF data stores.

The REST services can be divided into the following groups:

- **Server generic services:** allows access to available data sources, and configuration settings for general, proxy, and logging parameters.
- **Oracle RDF services:** allows CRUD operations on Oracle RDF objects.

- **SPARQL services:** allows execution of SPARQL queries and updates on the data sources.

Assuming the deployment of RDF web application with context-root set to `orardf`, on `localhost` machine and port number `7101`, the base URL for REST requests is `http://localhost:7101/orardf/api/v1`.

Most of the REST services are protected with Form-based authentication. Administrator users can define a public RDF data source using the RDF Graph Server and Query UI web application. The public REST endpoints will then be available to perform SPARQL queries on published datasets.

> **Note:**
>
> The examples in this section and throughout this chapter reference host machine as `localhost` and port number as `7101`. These values can vary depending on your application deployment.

The following are some RDF REST examples:

- Get the server information:
  The following is a public endpoint URL. It can be used to test if the server is up and running.

  ```
  http://localhost:7101/orardf/api/v1/utils/version
  ```

- Get a list of data sources:
  `http://localhost:7101/orardf/api/v1/datasources`
- Get general configuration parameters:
  `http://localhost:7101/orardf/api/v1/configurations/general`
- Get a list of RDF semantic networks for Oracle RDF:
  `http://localhost:7101/orardf/api/v1/networks?datasource=rdfuser_ds_193c`
- Get a list of all Oracle RDF real models for a private semantic network (applies from 19c databases):
  `http://localhost:7101/orardf/api/v1/models?datasource=rdfuser_ds_193c&networkOwner=RDFUSER&networkName=LOCALNET&type=real`
- Post request for SPARQL query:
  `http://localhost:7101/orardf/api/v1/datasets/query?datasource=rdfuser_ds_193c&datasetDef={"metadata":[ {"networkOwner":"RDFUSER", "networkName":"LOCALNET","models":["UNIV_BENCH"]} ] }`

  Query Payload: `select ?s ?p ?o where { ?s ?p ?o} limit 10`
- Get request for SPARQL query:
  `http://localhost:7101/orardf/api/v1/datasets/query?datasource=rdfuser_ds_193c&query=select ?s ?p ?o where { ?s ?p ?o} limit 10&datasetDef={"metadata":[ {"networkOwner":"RDFUSER", "networkName":"LOCALNET","models":["UNIV_BENCH"]} ] }`
- Put request to publish an RDF model:
  `http://localhost:7101/orardf/api/v1/datasets/publish/DSETNAME?datasetDef={"metadata":[ {"networkOwner":"RDFUSER", "networkName":"LOCALNET" "models":["UNIV_BENCH"]} ]}`

  Default SPARQL Query Payload: `select ?s ?p ?o where { ?s ?p ?o} limit 10`

This default SPARQL can be overwritten when requesting the contents of a published dataset. The `datasource` parameter in the preceding request is optional. However, if you define this parameter on the URL, it must match the current publishing data source name because this API version supports just one publishing data source. Otherwise, the published data source name is automatically used.

- Get request for a published dataset:
  The following is a public endpoint URL. It is using the default parameters (SPARQL query, output format, and others) that are stored in dataset definition. However, these default parameters can be overwritten in REST request by passing new parameter values.

  ```
  http://localhost:7101/orardf/api/v1/datasets/query/published/DSETNAME
  ```

A detailed list of available REST services can be found in the Swagger json file, `orardf_swagger.json`, which is packaged in the application documentation directory.

14

Oracle RDF Graph Query UI

The Oracle RDF Graph Query UI is an Oracle JET based client that can be used to manage RDF objects from different data sources, and to perform SPARQL queries and updates.

This Java EE application helps to build application webpages that query and display RDF graphs. It supports queries across multiple data sources.

- Installing RDF Graph Query UI
- Managing User Roles for RDF Graph Query UI
- Getting Started with RDF Graph Query UI
- Accessibility

14.1 Installing RDF Graph Query UI

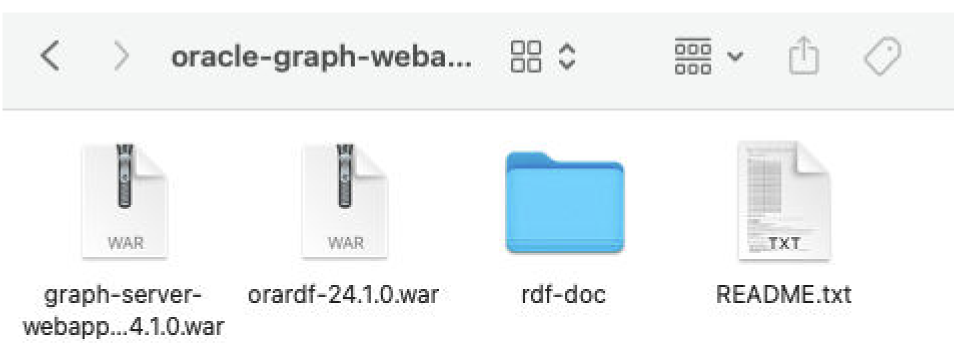
In order to get started on Oracle RDF Graph Query UI, you must download and install the application.

You can download RDF Graph Query UI using one of the following options:

- Download **Oracle Property Graph and Oracle RDF Graph Webapps** from Oracle Graph Server and Client Downloads page on Oracle Technology Network.
- Download the **Oracle Graph Webapps** component in Oracle Graph Server and Client deployment from Oracle Software Delivery Cloud.

The downloaded `oracle-graph-webapps-<version>.zip` deployment contains the files as shown in the following figure:

Figure 14-1 Oracle Graph Webapps deployment



The deployment of the RDF `.war` file provides the Oracle RDF Graph Query UI console.

> **Note:**
>
> Starting from Release 24.1.0, the RDF Graph Query UI web application is based on JDK 11. Therefore, ensure that the application servers (WebLogic or Tomcat) support JDK 11. In the case of the WebLogic server, use version 14.1.1.0.

The `rdf-doc` folder contains the User Guide documentation.

This deployment also includes the REST API running on the application server to handle communication between users and backend RDF data stores.

14.2 Managing User Roles for RDF Graph Query UI

Users will have access to the application resources based on their role level. In order to access the Query UI application, you need to enable a role for the user.

The following describes the different user roles and their privileges:

- **Administrator:** An administrator has full access to the Query UI application and can update configuration files, manage RDF objects and can execute SPARQL queries and SPARQL updates.
- **RDF:** A RDF user can read or write Oracle RDF objects and can execute SPARQL queries and SPARQL updates. But, cannot modify configuration files.
- **Guest:** A guest user can only read Oracle RDF objects and can only execute SPARQL queries.

Figure 14-2 User Roles for RDF Graph Query

Task	ADMIN	RDF	GUEST
Manage configuration settings	☑		
Manage Oracle RDF objects (read)	☑	☑	☑
Manage Oracle RDF objects (write)	☑	☑	
SPARQL query	☑	☑	☑
SPARQL update	☑	☑	

Application servers, such as WebLogic Server, Tomcat, and others, allow you to define and assign users to user groups. Administrators are set up at the time of the RDF Graph server installation, but the RDF and guest users must be created to access the application console.

- Managing Groups and Users in WebLogic Server
- Managing Users and Roles in Tomcat Server

14.2.1 Managing Groups and Users in WebLogic Server

The security realms in WebLogic Server ensures that the user information entered as a part of installation is added by default to the Administrators group. Any user assigned to this group will have full access to the RDF Graph Query UI application.

To open the WebLogic Server Administration Console, enter `http://localhost:7101/console` in your browser and logon using your administrative credentials. Click on Security Realms as shown in the following figure:

Figure 14-3 WebLogic Server Administration Console

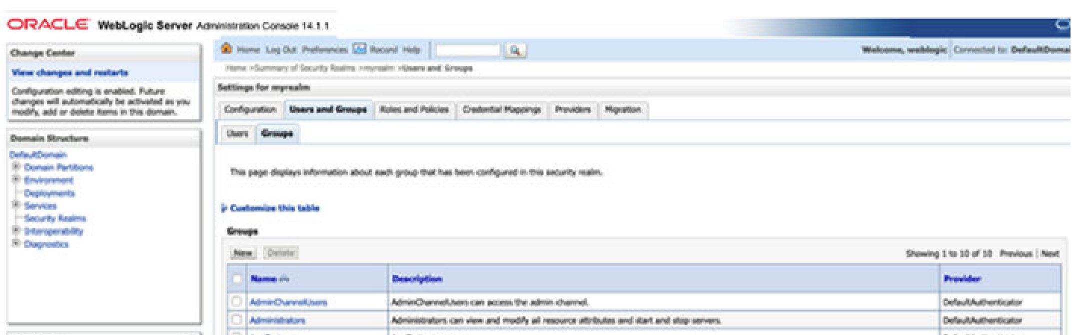
- Creating User Groups in WebLogic Server
- Creating RDF and Guest Users in WebLogic Server

14.2.1.1 Creating User Groups in WebLogic Server

To create new user groups in WebLogic Server:

1. Select the security realm from the listed Realms in Figure 14-3.
2. Click **Users and Groups** and then **Groups**.
3. Click **New** to create new RDF user groups in Weblogic as shown below:

Figure 14-4 Creating new user groups in WebLogic Server



The following example creates the following two user groups:

- **RDFreadUser**: for guest users with just read access to application.
- **RDFreadwriteUser**: for users with read and write access to RDF objects.

Figure 14-5   Created User Groups in WebLogic Server

☐	RDFreadUser	RDF users with just read access (get requests)	DefaultAuthenticator
☐	RDFreadwriteUser	Rdf users with read and write access (but no write to configuration files)	DefaultAuthenticator

New Delete   Showing 1 to 10 of 10 Previous | Next

14.2.1.2 Creating RDF and Guest Users in WebLogic Server

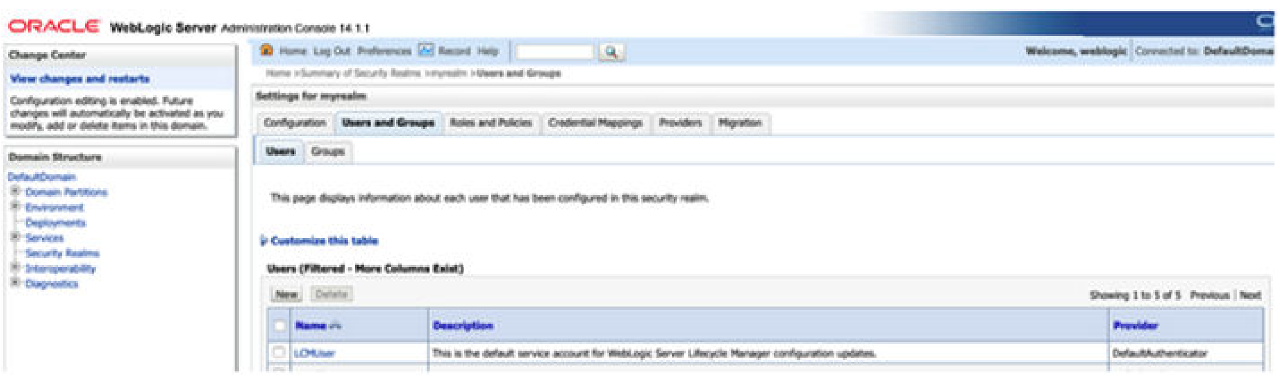
In order to have RDF and guest users in the user groups you must first create the RDF and guest users and then assign them to their respective groups.

To create new RDF and guest users in WebLogic server:

Prerequisites: RDF and guest users groups must be available or they must be created. See Creating User Groups in WebLogic Server for creating user groups.

1. Select the security realm from the listed Realms as seen in Figure 14-3
2. Click **Users and Groups tab** and then **Users**.
3. Click **New** to create the RDF and guest users.

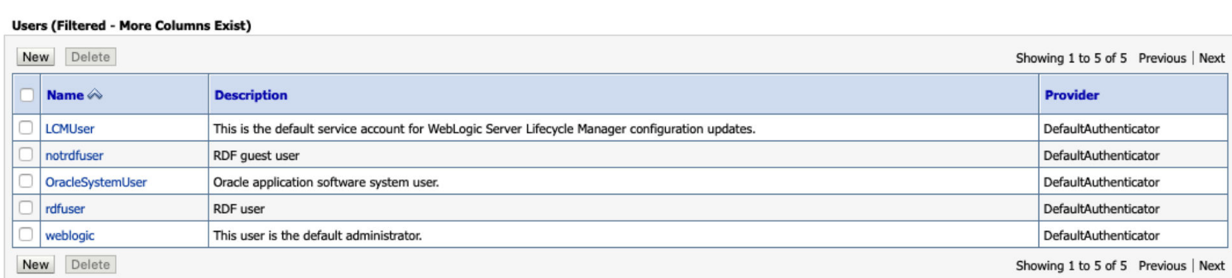
Figure 14-6   Create new users in WebLogic Server



The following example creates two new users :

- **rdfuser:** user to be assigned to group with read and write privileges.
- **nonrdfuser:** guest user to be assigned to group with just read privileges.

Figure 14-7   New RDF and Guest users



Users (Filtered - More Columns Exist)

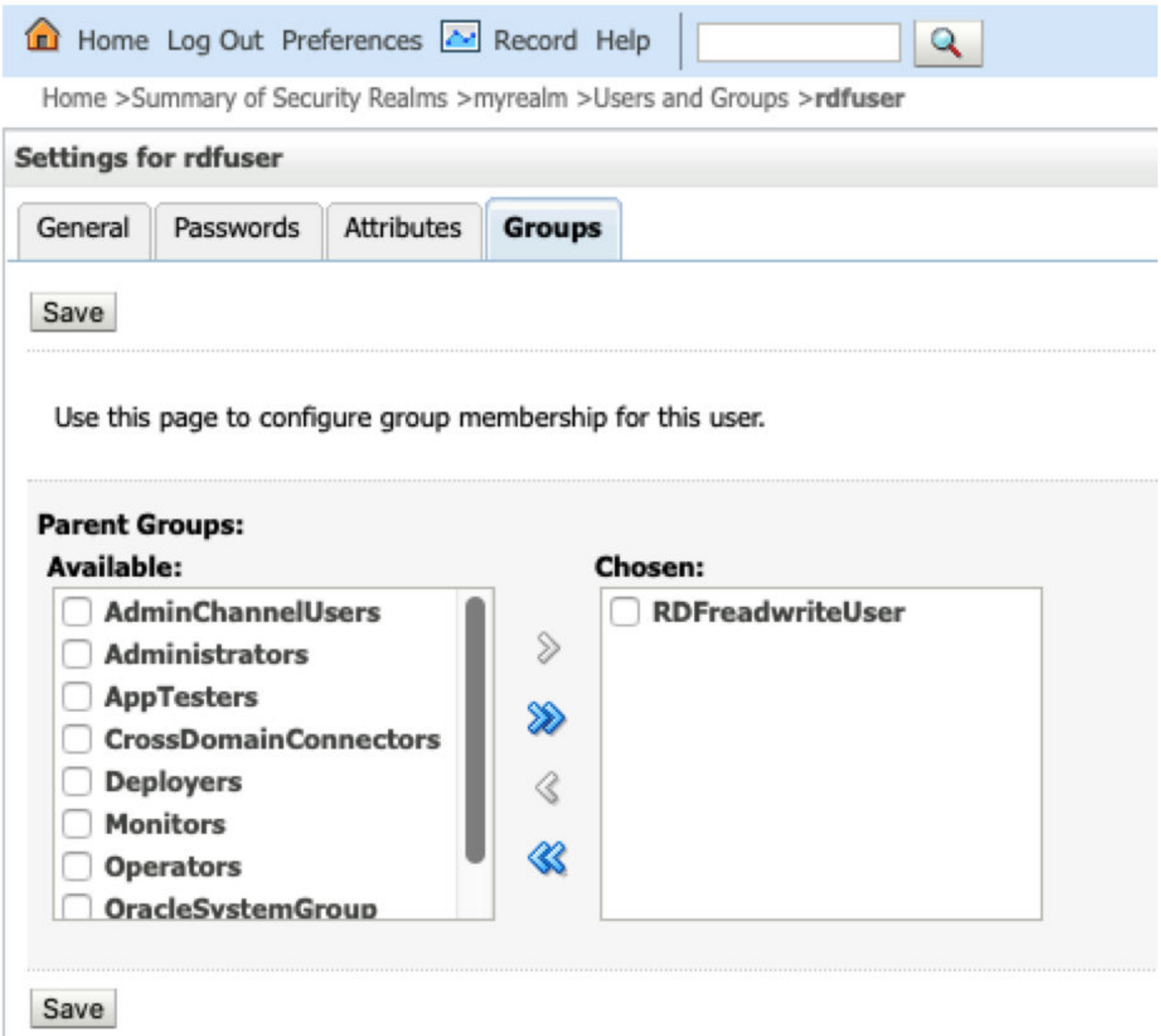
New Delete   Showing 1 to 5 of 5 Previous | Next

☐	Name	Description	Provider
☐	LCMUser	This is the default service account for WebLogic Server Lifecycle Manager configuration updates.	DefaultAuthenticator
☐	notrdfuser	RDF guest user	DefaultAuthenticator
☐	OracleSystemUser	Oracle application software system user.	DefaultAuthenticator
☐	rdfuser	RDF user	DefaultAuthenticator
☐	weblogic	This user is the default administrator.	DefaultAuthenticator

New Delete   Showing 1 to 5 of 5 Previous | Next

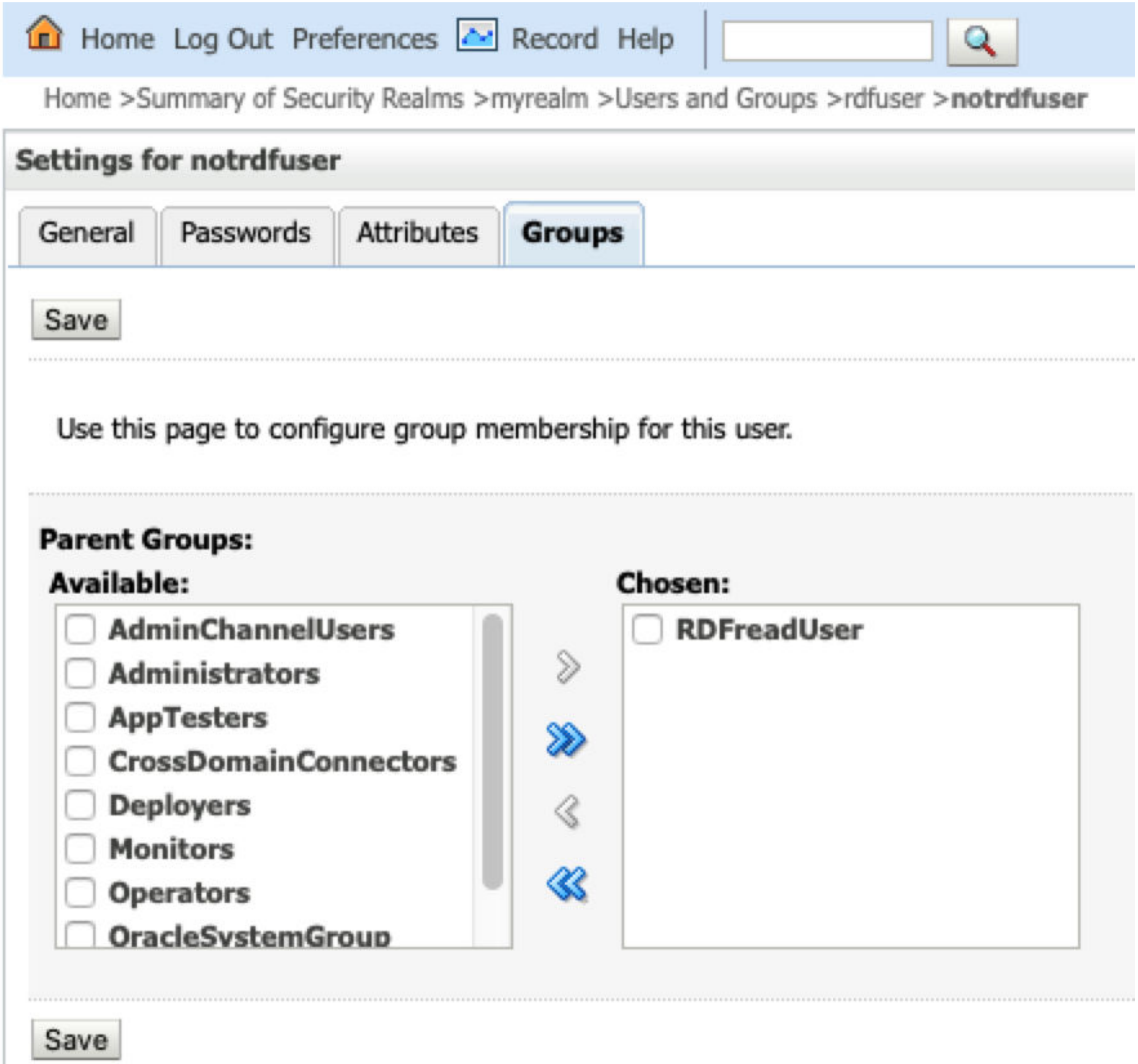
4. Select a user name and click **Groups** to assign the user to a specific group.
5. Assign **rdfuser** to **RDFreadwriteUser** group.

Figure 14-8 RDF User



6. Assign **nonrdfuser** to **RDFreadUser** group.

Figure 14-9 RDF Guest User



14.2.2 Managing Users and Roles in Tomcat Server

For Apache Tomcat, edit the Tomcat users file `conf/tomcat-users.xml` to include the RDF user roles. For example:

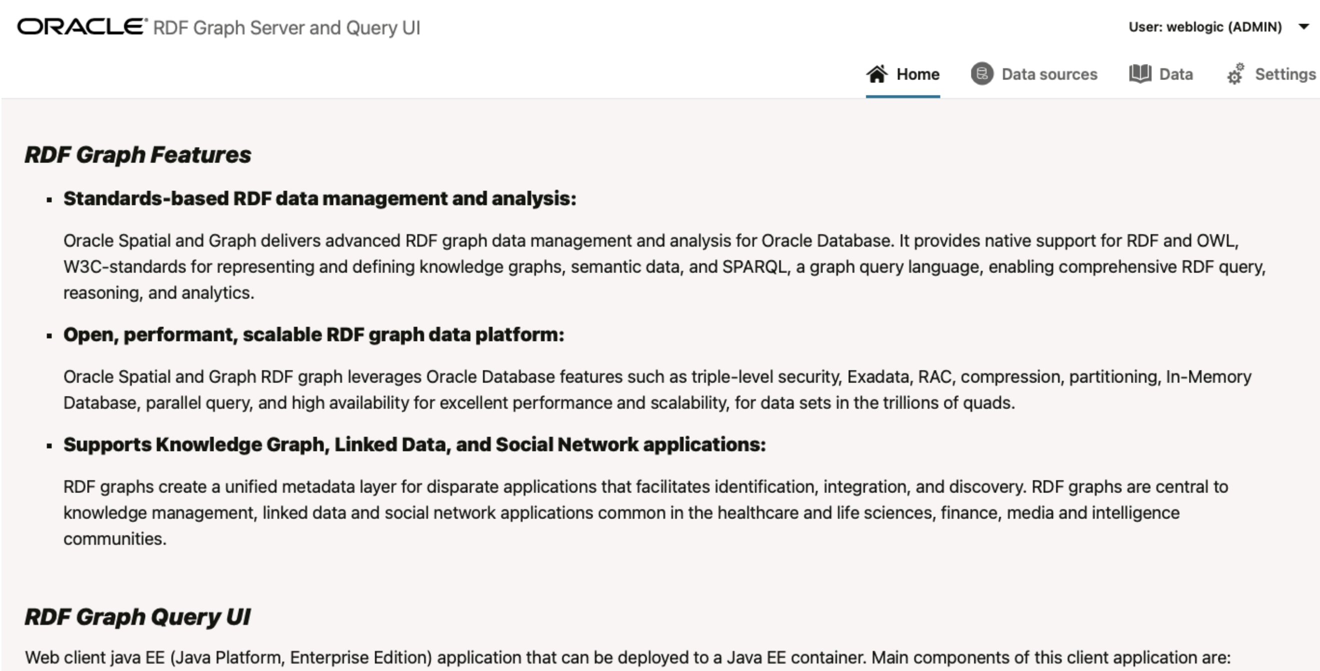
```
<tomcat-users xmlns="http://tomcat.apache.org/xml" xmlns:xsi="http://www.w3.org/2001/
XMLSchema-instance" version="1.0" xsi:schemaLocation="http://tomcat.apache.org/xml
tomcat-users.xsd">

    <role rolename="rdf-admin-user"/>

    <role rolename="rdf-read-user"/>

    <role rolename="rdf-readwrite-user"/>

    <user password="adminpassword" roles="manager-script,admin,rdf-admin-user"
username="admin"/>
```

```
    <user password="rdfuserpassword" roles="rdf-readwrite-user" username="rdfuser"/>

    <user password="notrdfuserpassword" roles="rdf-read-user" username="notrdfuser"/>

</tomcat-users>
```

14.3 Getting Started with RDF Graph Query UI

The Oracle Graph Query UI contains a main page with RDF graph feature details and links to get started.

Figure 14-10 Query UI Main Page



The main page includes the following:

- Home: Get an overview of the Oracle RDF Graph features.
- Data sources: Manage your data sources.
- Data: Manage, query or update RDF objects.
- Settings: Set your configuration parameters.

- Data Sources Page
- RDF Data Page
- Configuration Files for RDF Server and Client

14.3.1 Data Sources Page

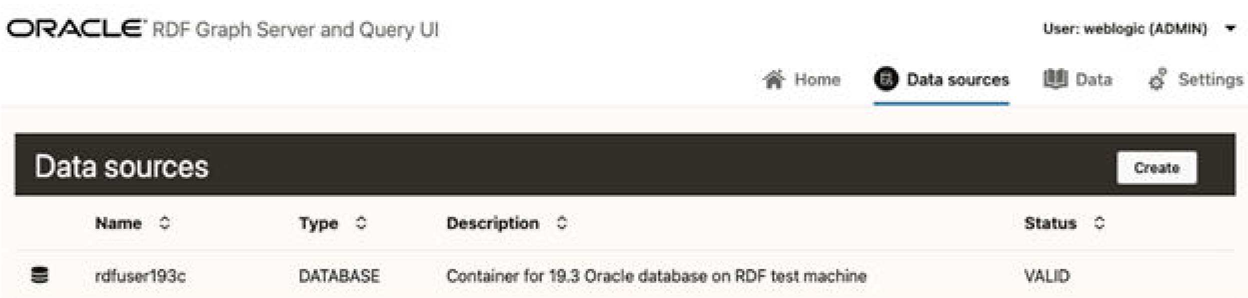
The Data Sources page allows you to create different types of data sources. Only administrator users can manage data sources. The RDF store can be linked to an Oracle Database or to an external RDF data provider. For Oracle data sources, there are three types of connections:

- JDBC data source defined with database parameters
- JDBC data source defined on an application server
- Oracle wallet connection defined in a zip file

These database connections must be available in order to link the RDF web application to the data source.

To create a data source, click **Data Sources**, then **Create**.

Figure 14-11 Data Sources Page



- Creating a JDBC URL Data Source
- Creating an Oracle Container Data Source
- Creating an Oracle Wallet Data Source
- Creating an Endpoint URL Data Source

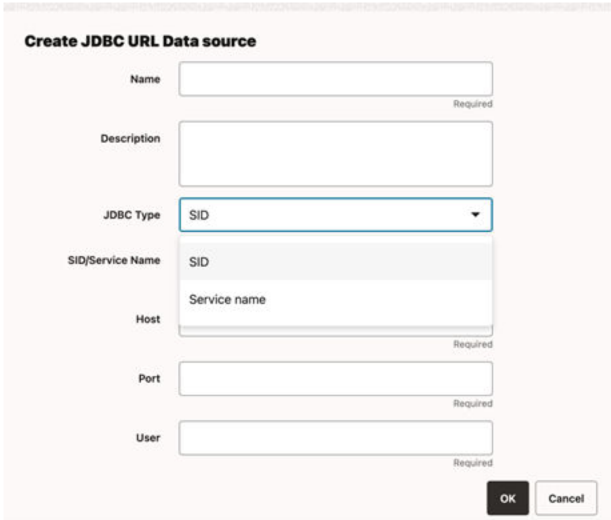
14.3.1.1 Creating a JDBC URL Data Source

Oracle JDBC URL is defined using the standard database parameters with user credentials. You can perform the following steps to create a JDBC URL data source:

1. Click **JDBC URL** in Figure 14-11.

   Create JDBC URL Data source dialog opens as shown:

Figure 14-12 Creating a JDBC URL Data Source



2. Enter the **Name** of the data source.
3. Optionally, enter **Description**.
4. Select the **JDBC Type**.
5. Enter **SID/Service Name** as appropriate.
6. Enter the **Host** and **Port** details.
7. Enter the **User** and **Password** credentials.
8. Click **OK** to create the data source.

14.3.1.2 Creating an Oracle Container Data Source

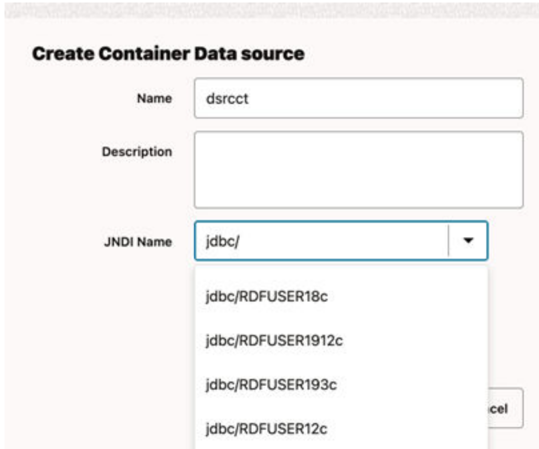
As a prerequisite to create a container data source in the RDF Graph Server and Query UI application, the JDBC data source must exist in the application server. See Creating a JDBC Data Source in WebLogic Server and Creating a JDBC Data Source in Tomcat for more information.

You can then perform the following steps to create an Oracle Container data source:

1. Click **Container** in Figure 14-11.

   Create Container Data source dialog opens as shown:

Figure 14-13 Create Container Data Source



2. Enter the **Name** of the data source.
3. Optionally, enter **Description**.
4. Select the **JNDI Name** that exists on the application server.
5. Click **OK** to create the data source.

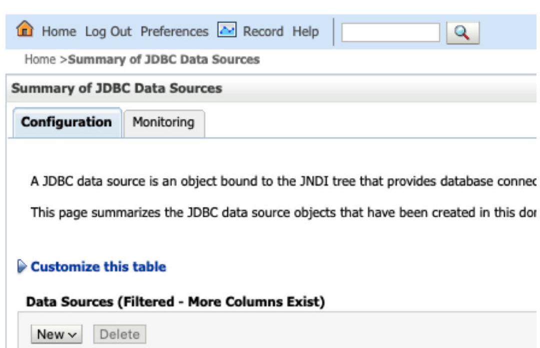
- Creating a JDBC Data Source in WebLogic Server
- Creating a JDBC Data Source in Tomcat

14.3.1.2.1 Creating a JDBC Data Source in WebLogic Server

To create a JDBC data source in WebLogic Server:

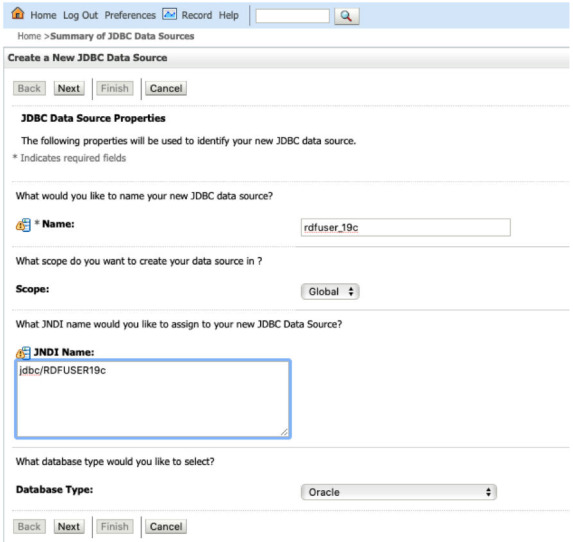
1. Log in to the **WebLogic administration console** as an administrator: `http://localhost:7101/console`.
2. Click **Services**, then **JDBC Data sources**.
3. Click **New** and select the **Generic data source** menu option to create a JDBC data source.

Figure 14-14 Generic Data Source



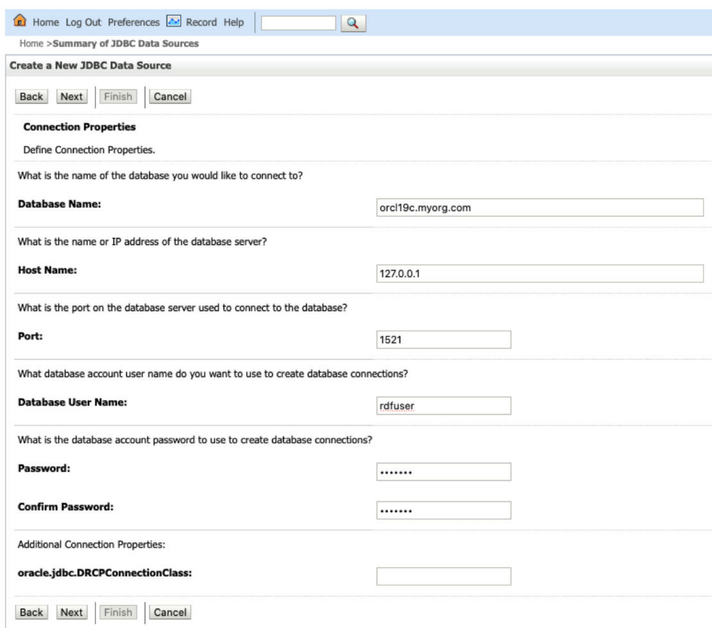
4. Enter the JDBC data source information (name and JNDI name), then click **Next**.

Figure 14-15   JDBC Data Source and JNDI



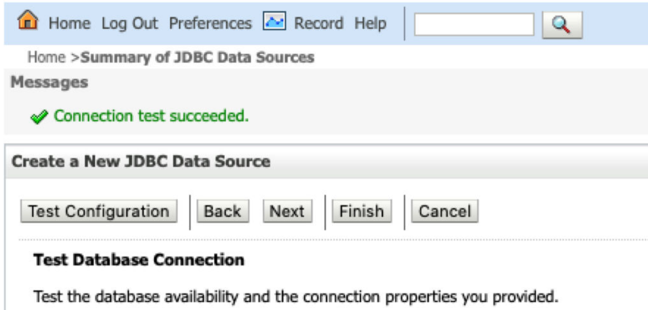
5. Accept the defaults on the next two pages.
6. Enter the database connection information: **service name**, **host**, **port**, and **user credentials**.

Figure 14-16   Create JDBC Data Source



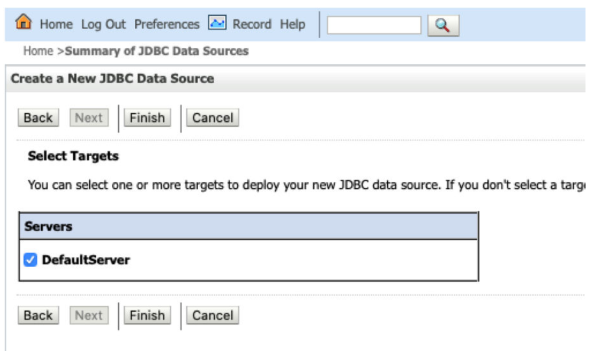
7. Click **Next** to continue.
8. Click the **Test Configuration** button to validate the connection and click **Next** to continue.

Figure 14-17 Validate connection



9. Select the **server target** and click **Finish**.

Figure 14-18 Create JDBC Data Source



The JDBC data gets added to the data source table and the JNDI name is added to the combo box list in the create container dialog.

14.3.1.2.2 Creating a JDBC Data Source in Tomcat

There are different ways to create a JDBC data source in Tomcat. See Tomcat documentaion for more details.

The following examples denote creation of JDBC data source in Tomcat by modifying the configuration files `conf/server.xml` and `conf/content.xml`.

- Add global JNDI resources on `conf/server.xml`.

```
<GlobalNamingResources>
    <Resource name="jdbc/RDFUSER19c" auth="Container" global="jdbc/RDFUSER19c"
              type="javax.sql.DataSource"
driverClassName="oracle.jdbc.driver.OracleDriver"
              url="jdbc:oracle:thin:@host.name:db_port_number:db_sid"
              username="rdfuser" password="rdfuserpwd" maxTotal="20" maxIdle="10"
              maxWaitMillis="-1"/>
 </GlobalNamingResources>
```

- Add the resource link to global JNDI’s on `conf/context.xml`:

```
<Context>
    <ResourceLink name="jdbc/RDFUSER19c"
                  global="jdbc/RDFUSER19c"
                  auth="Container"
                  type="javax.sql.DataSource" />
</Context>
```

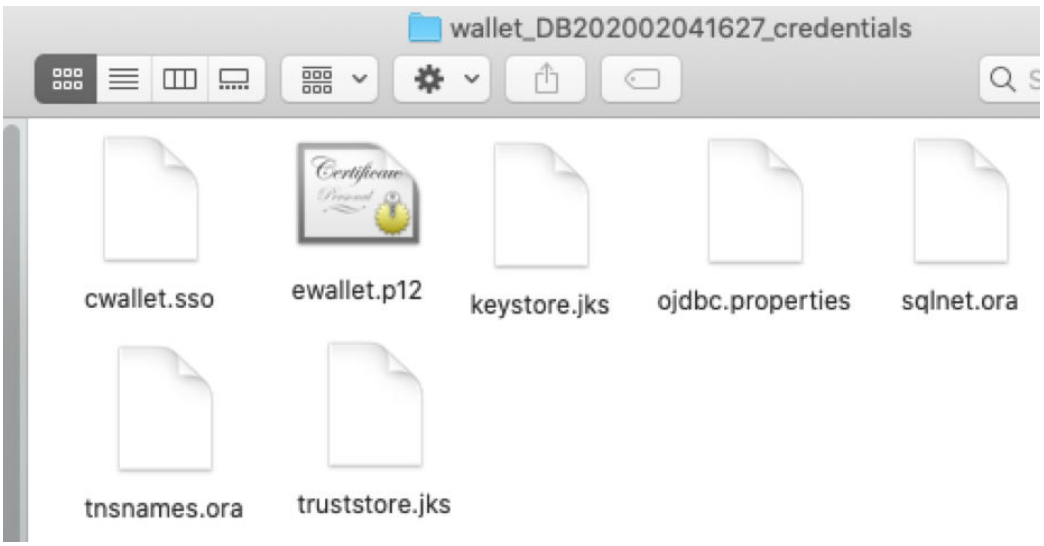
14.3.1.3 Creating an Oracle Wallet Data Source

To create a wallet data source in the Query UI application, you must have a wallet zip file. It can be a simple wallet zip file created with Oracle **orapki** utility, or a wallet downloaded from Oracle Autonomous Database.

In general, wallets are obtained from the Autonomous Database. See Download Client Credentials (Wallets) for more information to download a wallet from Oracle Autonomous Database.

The following figure displays the contents of the wallet zip file:

Figure 14-19 Cloud Wallet



Note that the *tnsnames.ora* file in the wallet zip file contains the wallet service alias names, and TCPS information. It does not contain the user credentials for each service.

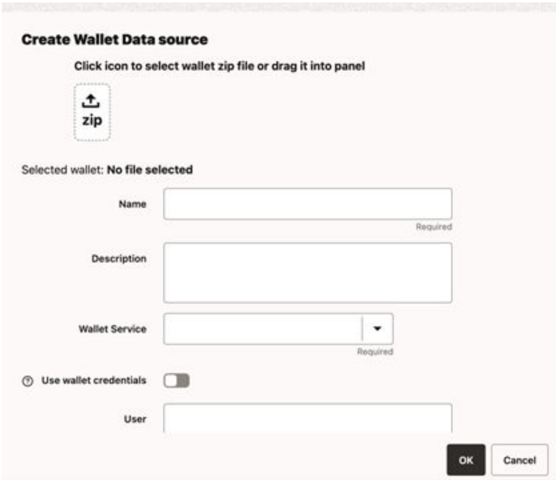
Using this wallet zip file, you can define an RDF wallet data source in the Query UI web application by directly entering the user credentials. Optionally, you can also have the user credentials stored inside the wallet for each desired service. If you choose to store the user credentials in the wallet, then see Storing User Credentials in a Wallet for more information.

The following describes the steps to create a wallet data source:

1. Click **Wallet** in Figure 14-11.

   Create Wallet Data source dialog opens as shown:

Figure 14-20 Wallet Data Source from cloud zip



2. Click the **upload** icon and select the wallet zip file.

   The zip file gets uploaded to the server.
3. Enter the data source **Name**.
4. Optionally, enter the **Description**.
5. Select the required **Wallet Service** name.
6. Provide the user credentials using one of the following options as it applies to you.
   - Enable **Use wallet credentials** if you have stored the user credentials in the wallet.
   - Otherwise, enter directly the **User** and **Password** credentials.
7. Optionally, enter the proxy details.

- Storing User Credentials in a Wallet

14.3.1.3.1 Storing User Credentials in a Wallet

The following steps describe the process for adding the credentials to the wallet zip file. It is important that you store this wallet file along with the credentials in a safe location for security reasons.

1. Unzip the **cloud wallet** zip file in a temporary directory.
2. Use the service name alias in the **tnsnames.ora** to store credentials by running the following command:
   For example, if the service name alias is *db202002041627_medium*:

```
${ORACLE_HOME}/bin/orapki secretstore create_credential -wallet /tmp/cloudwallet
           -connect_string db202002041627_medium -username username -password
password
```

3. Zip the cloud wallet files into a new zip file.

14.3.1.4 Creating an Endpoint URL Data Source

External data sources are connected to the RDF data store using the endpoint URL.

You can execute SPARQL queries and updates to the RDF data store using a base URL. In some cases, such as Apache Jena Fuseki, there are specific URLs based on the dataset name. For example:

- DBpedia Base URL: http://dbpedia.org/sparql
- Apache Jena Fuseki (assuming a dataset name `dset`):
  - Query URL: `http://localhost:8080/fuseki/dset/query`
  - Update URL: `http://localhost:8080/fuseki/dset/update`

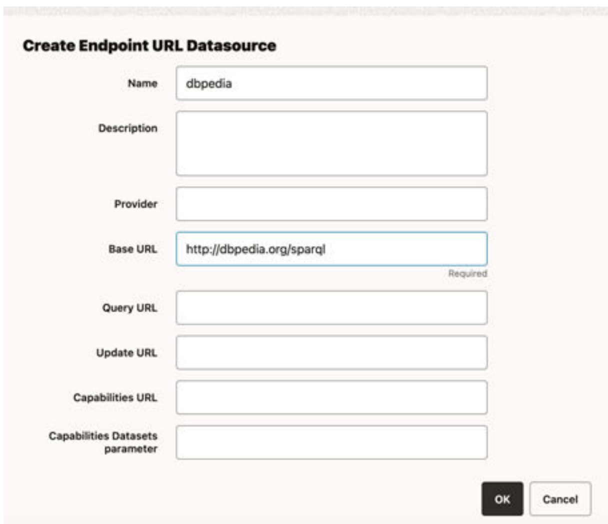
The RDF web application issues SPARQL queries to RDF datasets. These datasets can be retrieved from provider if a get capabilities request is available. For DBpedia, there is a single base URL to be used, and therefore a default single dataset is handled in application. For Apache Jena Fuseki, there is a request that returns the available RDF datasets in server: `http://localhost:8080/fuseki/$/server`. Using this request, the list of available datasets can be retrieved for specific use in an application.

You can perform the following steps to create an external RDF data source:

1. Click **Endpoint** in Figure 14-11.

   Create Endpoint URL Datasource dialog opens as shown. The following figure shows an example for creating a Dbpedia data source.

   Figure 14-21 DBpedia Data Source

2. Enter the **Name** of the data source.
3. Optionally, enter **Description**.
4. Optionally, enter the **Provider** name.
5. Enter the **Base URL** to access the RDF service.
6. Optionally, enter the **Query URL** to run SPARQL queries.

   Note that if the **Query URL** is not defined, then the **Base URL** is used.
7. Optionally, enter the **Update URL** to run SPARQL updates.

   Note that if the **Update URL** is not defined, then the **Base URL** is used.
8. Provide the **Capabilities Datasets parameter** properties to retrieve the dataset information from the RDF server.
9. Enter the **Get URL** address that should return a JSON response with information about the dataset.

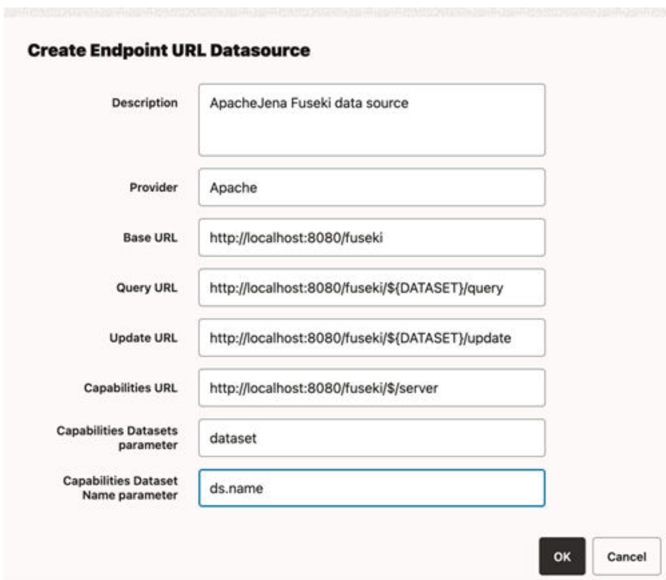
10. Enter the **Datasets parameter** property in JSON response that contains the dataset information.
11. Enter the **Dataset name parameter** property in datasets parameter that contains the dataset name.

> **Note:**
>
> For Jena Fuseki, the expression ${DATASET} will be replaced by the dataset name at runtime when SPARQL queries or SPARQL updates are being executed.

12. Click **OK** to create the data source.

    The following figure shows an example for creating an Apache Jena Fuseki data source.

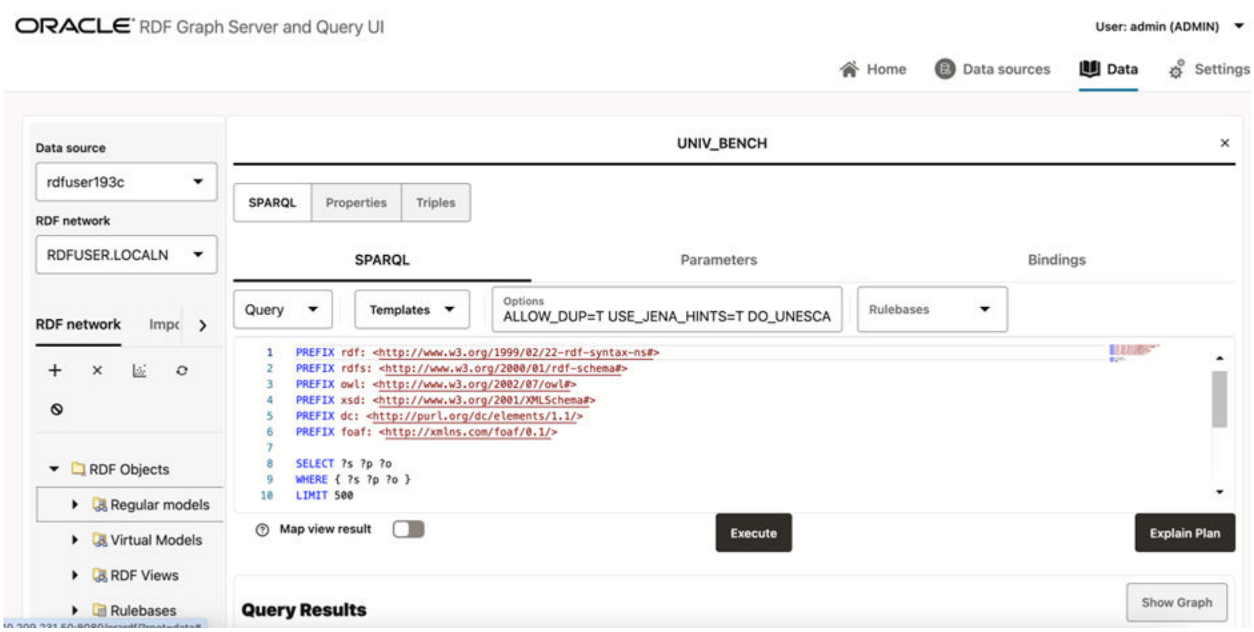
Figure 14-22 Apache Jena Fuseki Data Source



14.3.2 RDF Data Page

You can manage and query RDF objects in the RDF Data page.

Figure 14-23   RDF Data Page



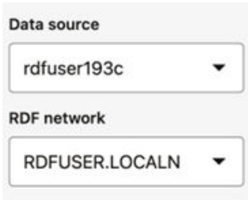
The left panel contains information on the available RDF data in the data source. The right panel is used for opening properties of a RDF object. Depending on the property type, SPARQL queries and SPARQL updates can be executed.

- Data Source Selection
- RDF Network Actions
- Importing Data
- SPARQL Query Cache Manager
- RDF Objects Navigator
- Data Source Published Datasets Navigator
- Performing SPARQL Query and SPARQL Update Operations
- Publishing Oracle RDF Models
- Published Dataset Playground
- Support for Result Tables
- Advanced Graph View
- Database Views from RDF Models

14.3.2.1 Data Source Selection

The data source can be selected from the list of available data sources present in Figure 14-11.

Figure 14-24 RDF Network

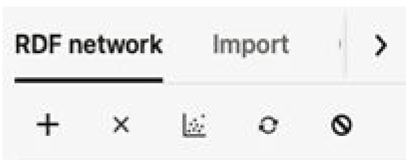


Select the desired Oracle RDF semantic network for the selected data source. Each network is identified by a network owner and network name.

14.3.2.2 RDF Network Actions

You can execute the following semantic network actions:

Figure 14-25 RDF Network Actions

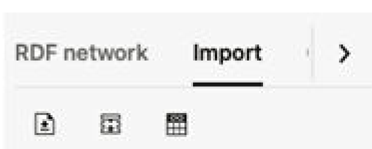


- Create an RDF network.
- Delete an RDF network.
- Gather statistics for a network.
- Refresh network indexes.
- Purge values not in use.

14.3.2.3 Importing Data

For Oracle RDF networks, the process of importing data into an RDF model is generally done by bulk loading the RDF triples that are available in the staging table.

Figure 14-26 RDF Import Data Actions



The available actions include:

- Upload one or more RDF files into a couple of Oracle RDF Graph staging tables. The staging table with suffix `_CLOB` will contain records with object values having length greater than 4k. These staging tables can be reused in other bulk load operations. Files with extensions `.nt` (N-triples), `.nq` (N-quads), `.ttl` (Turtle), `.trig` (TriG), and `.jsonld` (JsonLD) are supported for import. There is a limit of file size to be imported, which can be tuned by administrator.

Also, zip files can be used to import multiple files at once. However, the zip file is validated first, and will be rejected if any of the following conditions occur:

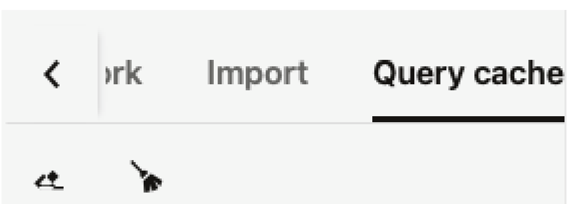
  – Zip file contains directories
  – Zip entry name extension is not a known RDF format (.nt, .nq, .ttl, .trig, .jsonld)
  – Zip entry size or compressed size is undefined
  – Zip entry size does exceed maximum unzipped entry size
  – Inflate ratio between compressed size and file size is lower than minimum inflate ratio
  – Zip entries total size does exceed maximum unzipped total size

- Bulk load the staging table records into an Oracle RDF model.
- View the status of bulk load events.

14.3.2.4 SPARQL Query Cache Manager

SPARQL queries are cached data source, and they apply to Oracle data sources. The translations of the SPARQL queries into SQL expressions are cached for Oracle RDF network models. Each model can stores up to 64 different SPARQL queries translations. The Query Cache Manager dialog, allows you to browse data source network cache for queries executed in models.

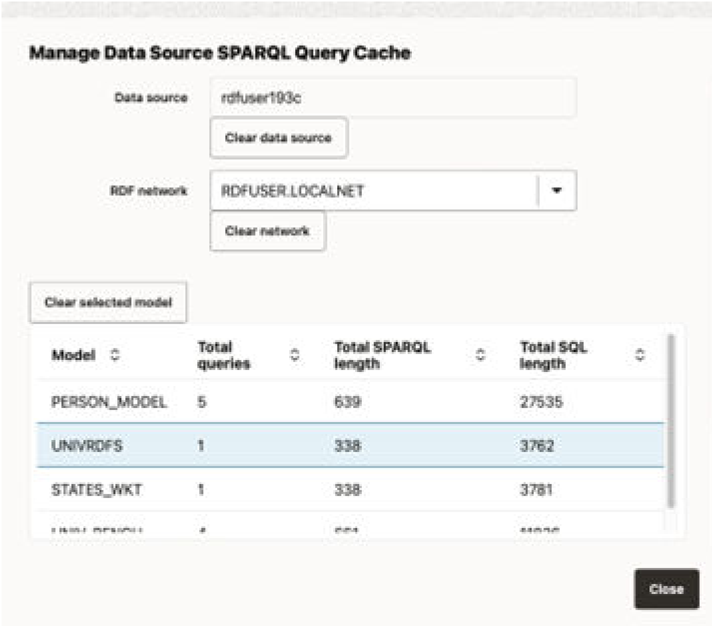
Figure 14-27 SPARQL Query Cache Manager



You can clear cache at different levels. The following describes the cache cleared against each level:

- **Data source:** All network caches are cleared.
- **Network:** All model caches are cleared.
- **Model**: All cached queries for model are cleared.
- **Model Cache Identifier:** Selected cache identifier is cleared.

Figure 14-28 Manage SPARQL Query Cache

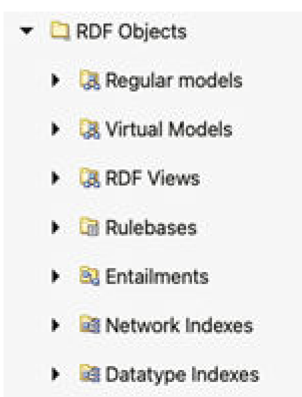


14.3.2.5 RDF Objects Navigator

The navigator tree shows the available RDF objects for the selected data source.

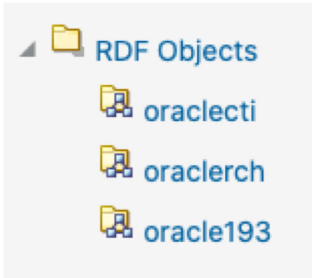
- For Oracle data sources, it will contain the different concept types like models, virtual models, view models, RDF view models, rule bases, entailments, network indexes, and datatype indexes.

Figure 14-29 RDF Objects for Oracle Data Source



- For endpoint RDF data sources, the RDF navigator will have a list of names representing the available RDF datasets in the RDF store.

Figure 14-30 RDF Objects from capabilities



- If an external RDF data source does not have a capabilities URL, then just a default dataset is shown.

Figure 14-31 Default RDF Object

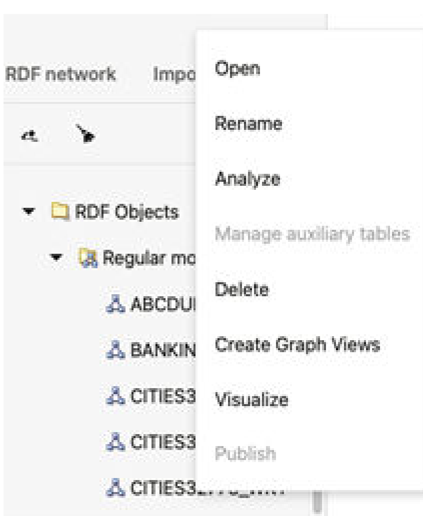
To execute SPARQL queries and SPARQL updates, open the selected RDF object in the RDF objects navigator. For Oracle RDF objects, SPARQL queries are available for models (regular models, virtual models, and view models).

Different actions can be performed on the navigator tree nodes. Right-clicking on a node under RDF objects will bring the context menu options (such as Open, Rename, Analyze, Manage auxiliary tables, Delete, Create Graph Views, Visualize, and Publish) for that specific node.

It is important to note the following:

- Publish menu item will be enabled only if the selected RDF data source is public.
- Guest users cannot perform actions that require a write privilege.

Figure 14-32 RDF Navigator - Context Menu



14.3.2.6 Data Source Published Datasets Navigator

If the selected RDF data source is public, a navigator node with the public datasets is displayed on the menu tree as shown in the following figure:

Figure 14-33 Data Source Published Datasets Navigator

14.3.2.7 Performing SPARQL Query and SPARQL Update Operations

To execute SPARQL queries and updates, open the selected RDF object in the RDF objects navigator. For Oracle RDF objects, SPARQL queries are available for regular models, virtual models, and view models.

You can define the following parameters for SPARQL queries:

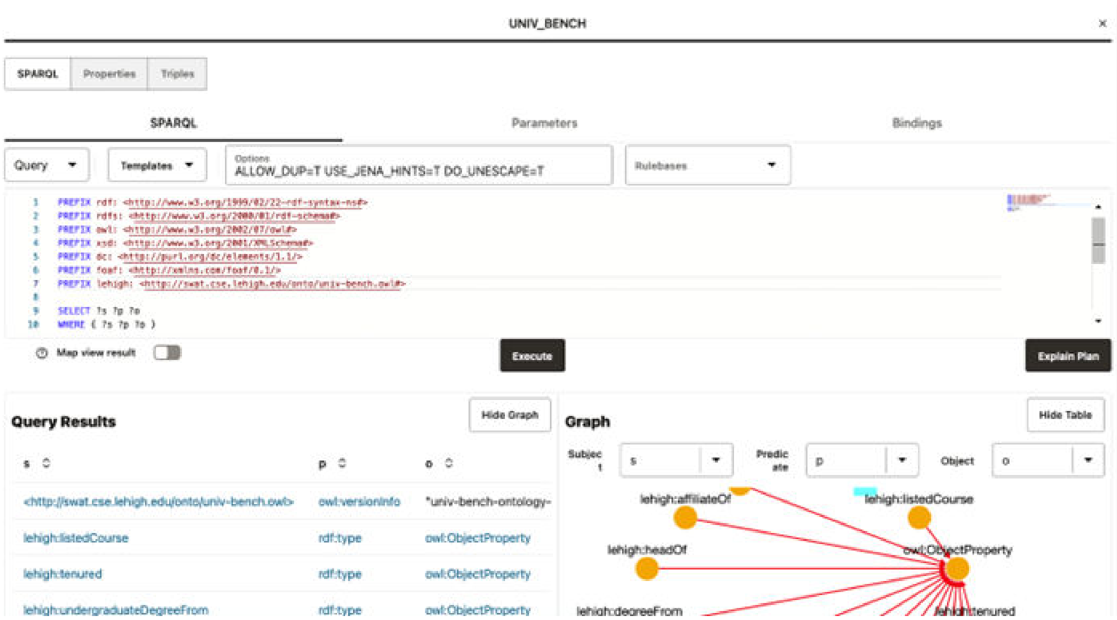
- **SPARQL:** The query string.
- **RDF options:** Oracle RDF options to be used when processing a query (See Additional Query Options for more information.).
- **Runtime parameters:** Fetch size, query timeout and others (this is applied to Oracle RDF data sources).
- **Rulebases:** Rulebase names associated with RDF model in an entailment. If none, then the selection box will be empty.
- **Binding parameters:** The expression `?ora__bind` is used as a binding parameter in a SPARQL string. Each binding parameter is defined by a type (uri or literal) and a value. For example:

  ```
  SELECT ?s ?p ?o WHERE { ?s ?p ?ora__bind } LIMIT 500
  ```

  An example of JSON representation of a binding parameter that can be passed to a REST query service is: `{ "type" : "literal", "value" : "abcdef" }`

The following figure shows the SPARQL query page, containing the graph view.

Figure 14-34 SPARQL Query Page

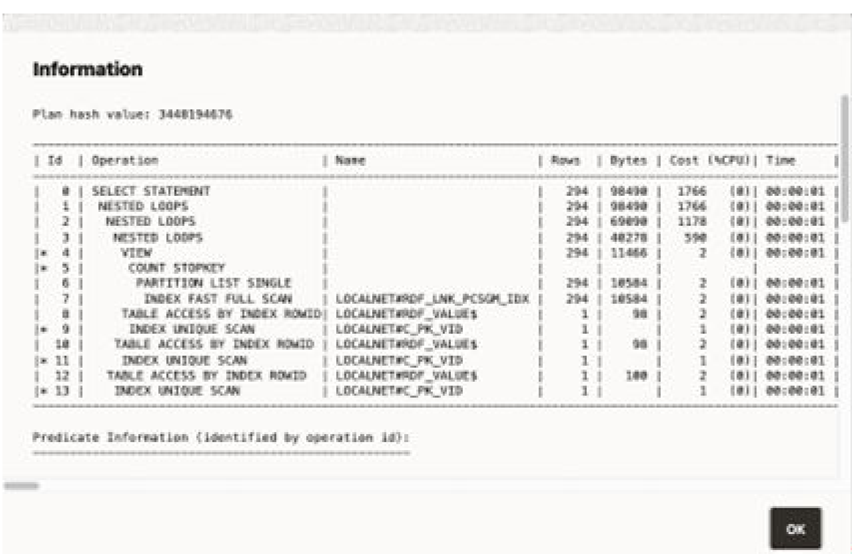


The number of results on the SPARQL query is determined by the limit parameter in SPARQL string, or by the maximum number of rows that can be fetched from server. As an administrator you can set the maximum number of rows to be fetched in the settings page.

A graph view can be displayed for the query results. On the graph view, you must map the columns for the triple values (subject, predicate, and object). In a table view, the columns that represent URI values have hyperlinks.

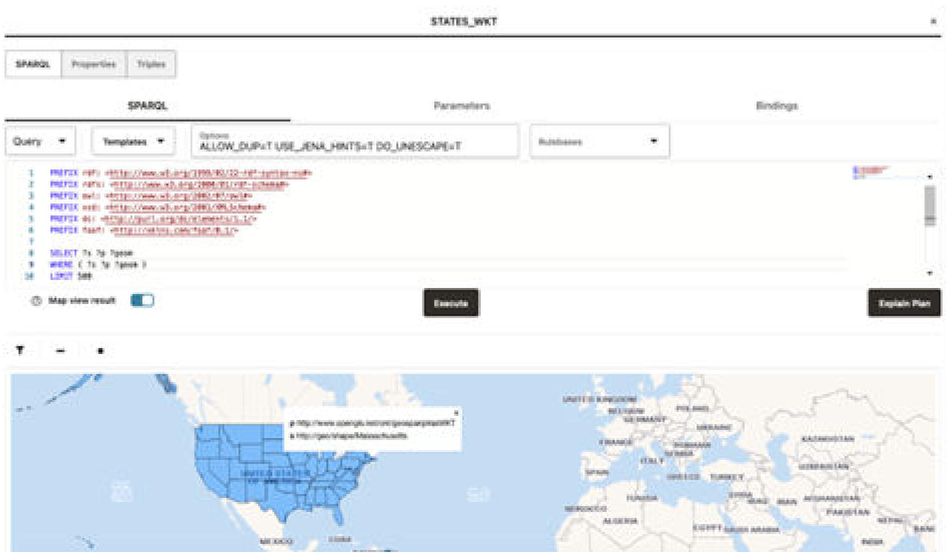
Besides the **Execute** button to run the SPARQL query, there is also the **Explain Plan** button to retrieve the SQL query plan for the SPARQL. This basically displays a dialog with the EXPLAIN PLAN results and the SPARQL translation.

Figure 14-35 SQL EXPLAIN PLAN for SPARQL Translation



For Oracle data sources, if the SPARQL query selects an RDF object value that represents a GeoSPARQL data type (such as `WKT`, `GML`, `KML`, or `GeoJSON`), a map visualization can be displayed by switching on **Map view result**. In this case, the SPARQL query must select the geometry attribute which is an RDF literal of a GeoSPARQL data type. On execution, this query will produce a GeoJSON result which is then passed to the map component for visualization. For example:

Figure 14-36 Map Visualization for GeoSPARQL Data Types in a SPARQL Query



14.3.2.8 Publishing Oracle RDF Models

Oracle RDF models can be published as datasets. These are then available through a public REST endpoint for SPARQL queries. Administrator users can define a public RDF data source for publishing data by configuring the application general parameters (see General JSON configuration file).

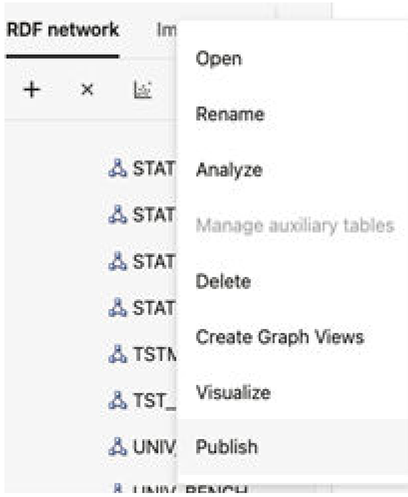
> **Note:**
>
> It is important to be aware that by enabling RDF data publishing and defining a public RDF data source, your public URL endpoints for RDF datasets are exposed. This endpoint URL can be used directly in applications without entering credentials.

However, public endpoints have some security constraints related to execution of SPARQL queries. SPARQL updates, SPARQL SERVICE, and SPARQL user-defined functions are not allowed.

To publish an Oracle RDF model as a dataset:

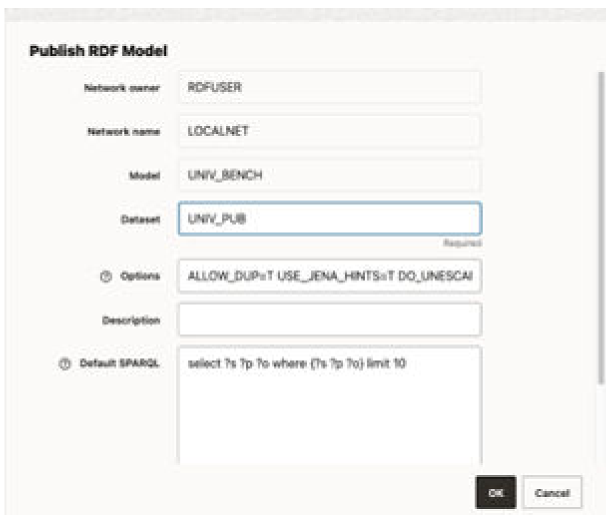
1. Right-click on the RDF model and select **Publish** from the menu as shown:

Figure 14-37 Publish Menu



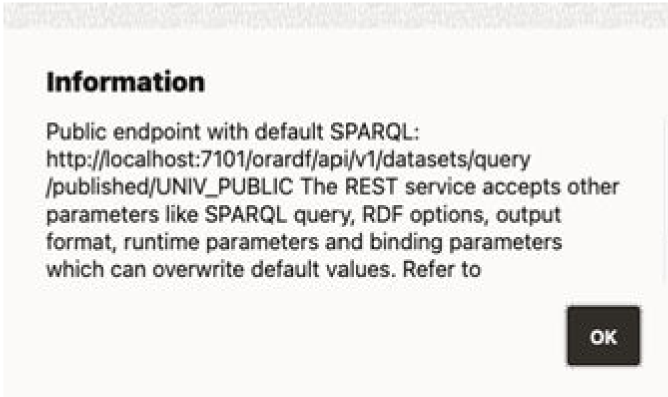
2. Enter the **Dataset** name (mandatory), **Description**, and **Default SPARQL**. This default SPARQL can be overwritten on the REST request.

Figure 14-38 Publish RDF Model



3. Click **OK**.
The public endpoint GET URL for the dataset is displayed. Note that the POST request can also be used to access the endpoint.

Figure 14-39 GET URL Endpoint



This URL uses the default values defined for the dataset and follows the pattern shown:

```
http://${hostname}:${port_number}/orardf/api/v1/datasets/query/published/$
{dataset_name}
```

You can override the default parameters stored in the dataset by modifying the URL to include one or more of the following parameters:

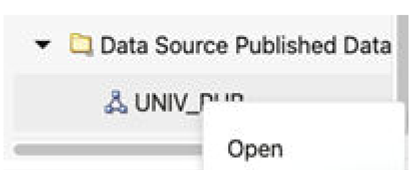
- **query:** SPARQL query
- **format:** Output format (`JSON`, `XML`, `CSV`, `TSV`, `GeoJSON`, `N-Triples`, `Turtle`)
- **options:** String with Oracle RDF options
- **rulebases:** Rulebase names associated with dataset RDF model in an entailment
- **params:** JSON string with runtime parameters (`timeout`, `fetchSize`, and others)
- **bindings:** JSON string with binding parameters (URI or literal values)

The following shows the general pattern of the REST request to query published datasets (assuming the context root as `orardf`):

```
http://${hostname}:${port_number}/orardf/api/v1/datasets/query/published/$
{dataset_name}?datasource=${datasource_name}&query=${sparql}&format=$
{format}&options=${rdf_options}&params=${runtime_params}&bindings=$
{binding_params}
```

In order to modify the default parameters, you must open the RDF dataset definition by selecting **Open** from the menu options shown in the following figure or by double clicking the published dataset:

Figure 14-40 Open an RDF Dataset Definition



The RDF dataset definition for the selected published dataset opens as shown:

Figure 14-41   RDF Dataset Definition

You can update the default parameters and preview the results.

> **Note:**
>
> - RDF user with administrator privileges can update and unpublish any dataset.
> - RDF user with read and write privileges can only manage the datasets that the user created.
> - RDF user with read privileges can only query the dataset.

14.3.2.9 Published Dataset Playground

You can explore the published RDF datasets from a public web page.

You can access the page using the following URL format:

```
{protocol}://{host}:{port}/{app_name}/public.html
```

For example:

```
http://localhost:7101/orardf/public.html
```

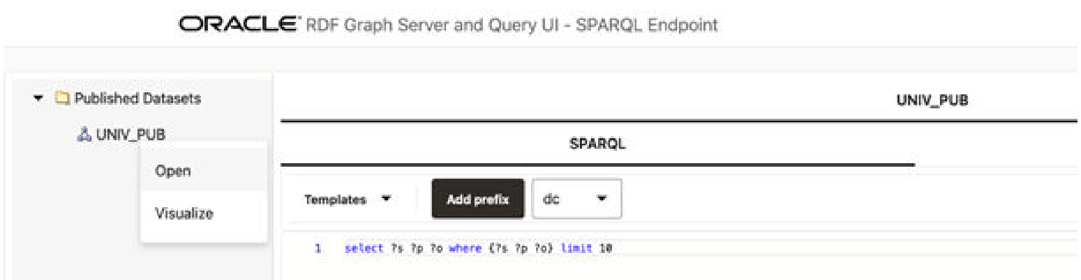
The public web page is displayed as shown:

Figure 14-42 Public Web Page

The main components of this public page are:

- **Published Datasets:** contains the names of the published RDF datasets for public RDF data source. To open the RDF dataset double click it or right click the tree dataset and execute the Open menu item as shown:

  Figure 14-43 Opening a Published Dataset on the Public Page

- The tab panel on the right allows you to execute SPARQL queries against the published RDF dataset. SPARQL query results are displayed in tabular as well as graph view formats. However, if the **Accessibility** switch on the top right corner of the page is switched `ON`, then the results are only displayed in tabular format.
  The following options are supported in the tab panel:
  - **Templates:** SPARQL template queries to use.
  - **Add prefix:** click to add the selected prefix in the combo box to a SPARQL query.
  - **SPARQL:** enter the SPARQL to be executed in the text area.
  - **select/ask:** select the output format for SPARQL `SELECT` and SPARQL `ASK` queries.
  - **construct/describe:** select the output format for SPARQL `CONSTRUCT` and SPARQL `DESCRIBE` queries.
  - **Execute:** click this button to execute the SPARQL query against the RDF public endpoint.
  - **Table:** shows the result in a tabular format.
  - **Raw:** shows the raw SPARQL result on specified format returned from server.

– **Download:** click to download the raw response.

14.3.2.10 Support for Result Tables

Result tables (also known as Subject-Property-Matrix (SPM) auxiliary tables) can be used to speed up SPARQL query execution. It is recommended you first refer to Speeding up Query Execution with Result Tables, for a detailed description of result tables. These auxiliary tables are associated with individual RDF graphs. Once they are created, they are automatically used during execution of SPARQL queries, unless specific options (see SPARQL Query Options for Result Tables) are passed in to indicate otherwise (note that the query cache may need to be cleared if the same SPARQL query is to be executed right after creating a result table).

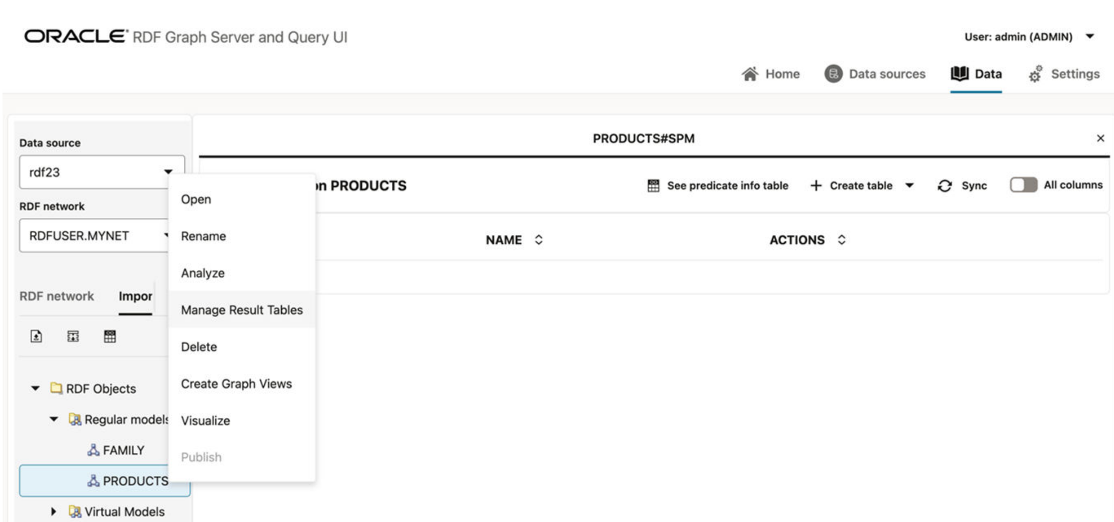
There are three types of result tables based on the type of the query pattern used for producing the results to be stored:

- Star-Pattern tables (also known as Single-Valued Property or SVP tables) hold values for single-valued RDF properties. A property `p` is single-valued in an RDF model if each resource in the model has at most one value for `p`.
- Triple-Pattern tables (also known as Multi-Valued Property (MVP) tables) hold values for individual RDF properties.The property used for a triple-pattern table may be single-valued or multi-valued. (A property p is multi-valued in an RDF graph if it contains two triples `(s p o1)` and `(s p o2)` with `o1` not equal to `o2`.)
- Chain-Pattern tables (also known as Property Chain (PCN) tables) hold paths in the RDF graph. A sequence of triples form a path if for each consecutive pair of triples, `Ti` and `Tj`, the object value of `Ti` is equal to the subject value of `Tj`.

Star-pattern and chain-pattern tables can be used to reduce joins during SPARQL query execution, while triple-pattern tables allow for more compact representation for triples involving individual properties. Additionally, if lexical values are included in the result tables, then joins needed for looking up lexical values can be avoided as well.

The RDF Server and Query UI web application provides support for creation and management of result tables. You can manage these auxiliary tables by right clicking the RDF graph and selecting the **Manage Result Tables** menu item as shown:

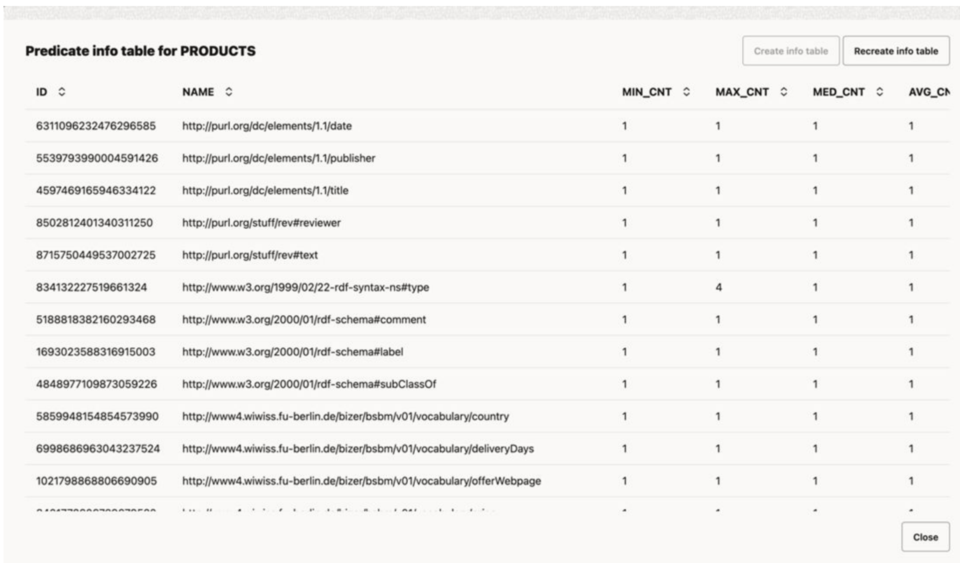
Figure 14-44 Result Tables



The **Result Tables** page displays a table which lists the result tables present in the RDF graph. If you are a first time user, then this table list will be empty. In such a case, you need to create the **Predicate Info Table** that is required for creating and managing the result tables. The predicate info table stores information about each property used as predicate in the RDF graph. For each property (or its inverse), the stored information includes its `id` (or negative `id`, for the inverse), `name`, and statistics on cardinality (that is, the number of triples that use this property) per distinct subject (or, object, for the inverse). A value of one in the `MAX_CNT` column indicates that the property was single-valued when the statistics were last computed.

To create the predicate info table, click **See predicate info table** in the table menu bar and then select **Create info table**. Note that you also have the option to recreate the table at a later time as shown in the following image:

Figure 14-45 Predicate Info Table



Once the predicate info table is created, you can then create and manage the result tables.

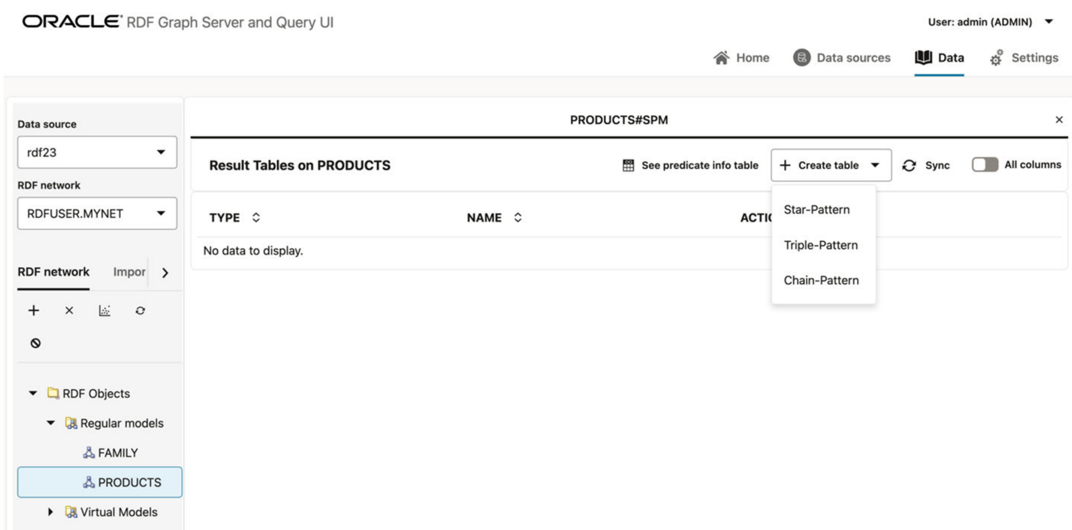
- Creating Result Tables
- Managing Result Tables

14.3.2.10.1 Creating Result Tables

You can create a result table by performing the following steps:

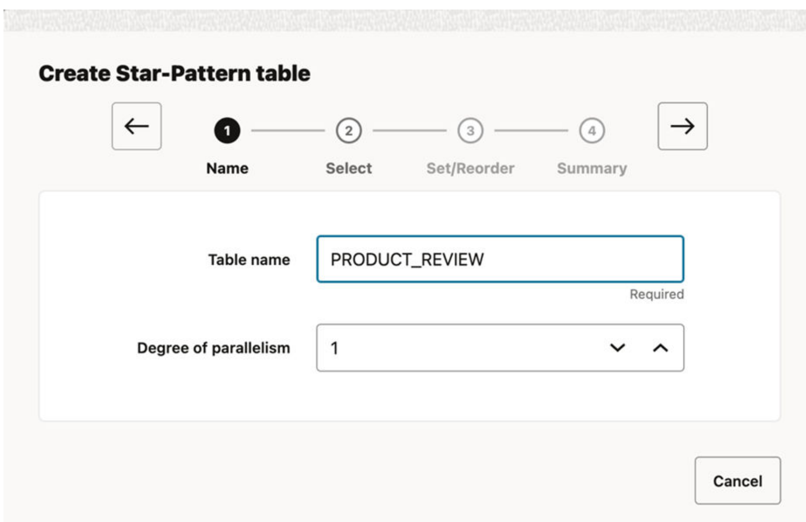
1. Click the **+ Create table** menu in the **Result Tables** menu bar, and select the type of the result table to be created.

Figure 14-46 Select the Type of Result Table



The **Create table** wizard opens and displays the first (**Name**) step of the workflow as shown:

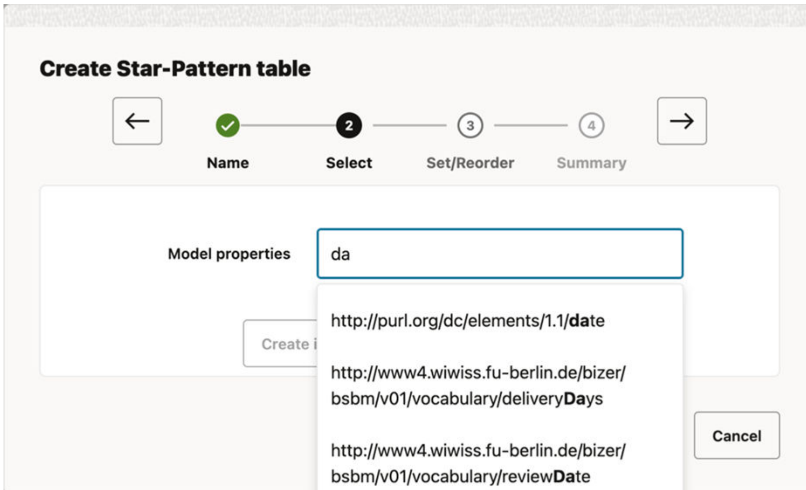
Figure 14-47 Step1: Name of the Result Table



2. Enter the **Table name**.
3. Optionally, select the **Degree of parallelism**.
4. Click → to move to the next step.

   The second (**Select**) step of the workflow opens as shown:

Figure 14-48 Step2: Select the Properties



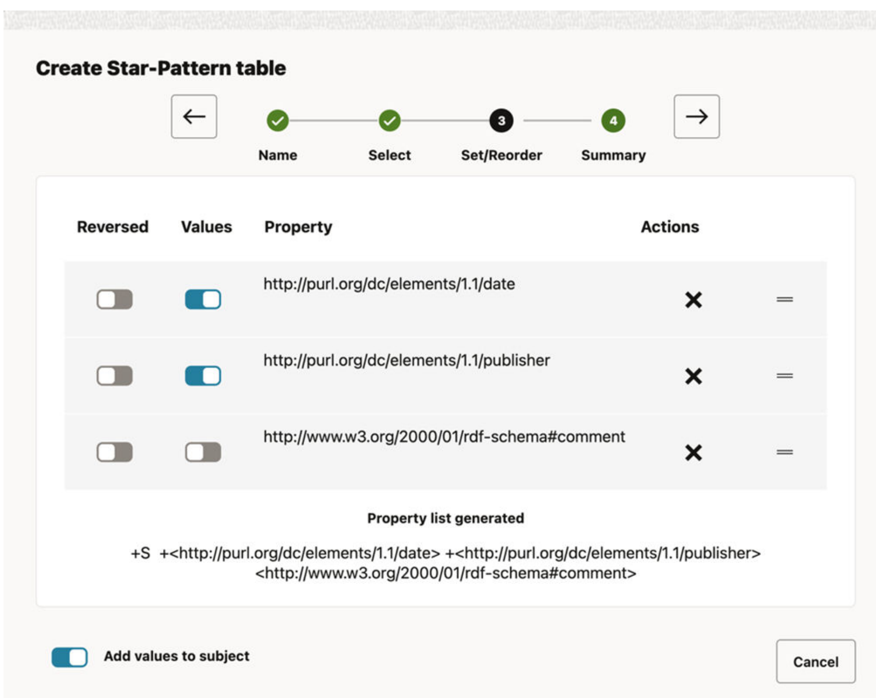
5. Select the result table properties from the **Model properties** dropdown or alternatively, you can type in a custom property.

   The dropdown contains all the properties present in the RDF graph.

6. Click → to move to the next step.

   The third (**Set/Reorder**) step of the workflow opens as shown:

Figure 14-49 Step3: Reorder and Configure Properties



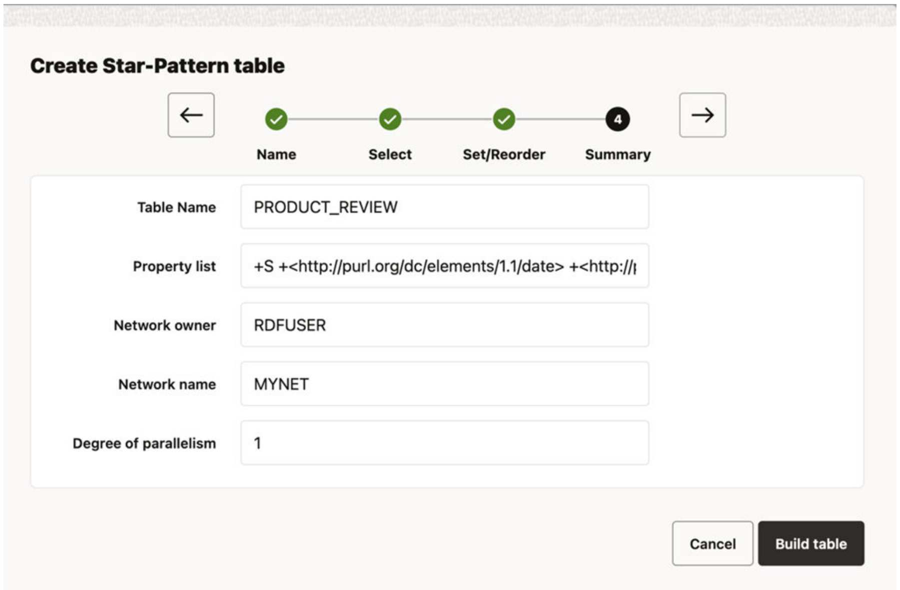
7. Drag and drop each property to reorder them as desired, include or exclude value columns and optionally configure any property inverse (not available for triple-pattern tables).

   This step leverages the all new complete flexibility present in 23ai to create your auxiliary table as desired.

8. Click → to move to the next step.

   The final (**Summary**) step of the workflow opens as shown:

Figure 14-50 Step4: Review the Selections

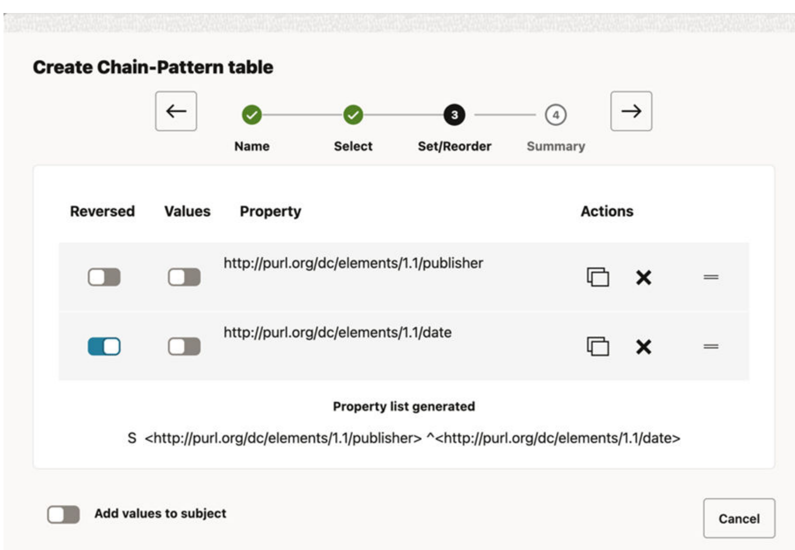


9. Review your selections and click **Build table** to create the result table.

The preceding workflow steps are common when creating any type of result table. However, the following 23ai features are available only for certain types of tables:

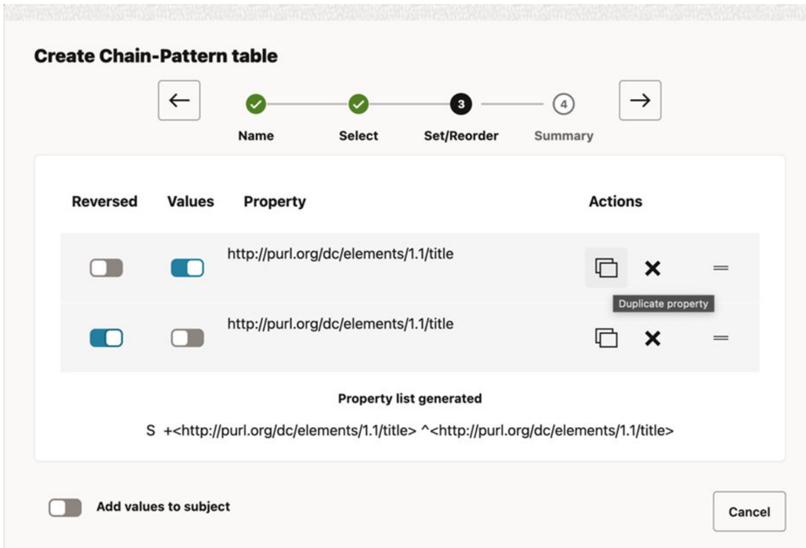
- **Inverse Property Path**: This feature is only available for chain-pattern or star-pattern tables. It can be enabled individually for each property, directly from the **Set/Reorder** properties step. In the following example, the property `http://purl.org/dc/elements/1.1/date` is shown **Reversed**. This implies that the subject and object of the triple switch places. See the W3C documentation for more information on inverse property path.

Figure 14-51 Configuring Inverse property path



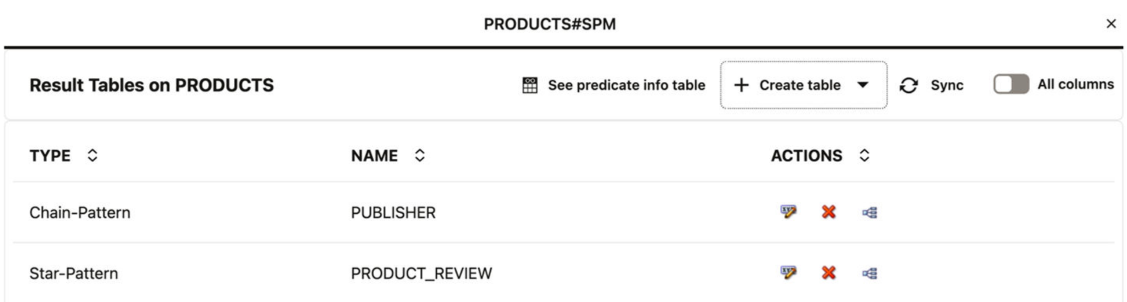
- **Multi-Occurrence** : This new feature in which the same property can be replicated multiple times is only available for chain-pattern tables. To duplicate a property, click the **Duplicate property** button under the **Actions** column on the **Set/Reorder** step. Note that once the table is created, for ease of differentiation, each replicated property is appended an identifier consisting of a “#” followed by a cardinal number.

Figure 14-52 Configuring Multi-Occurrence



14.3.2.10.2 Managing Result Tables

You can manage all your result tables directly from the **Result Tables** page which displays the list of the result tables. You can edit or delete result tables, and also view or create indexes on individual result tables. Make sure the result table view is set to **Tables only** to enable the **Actions** column, which displays the supported actions:

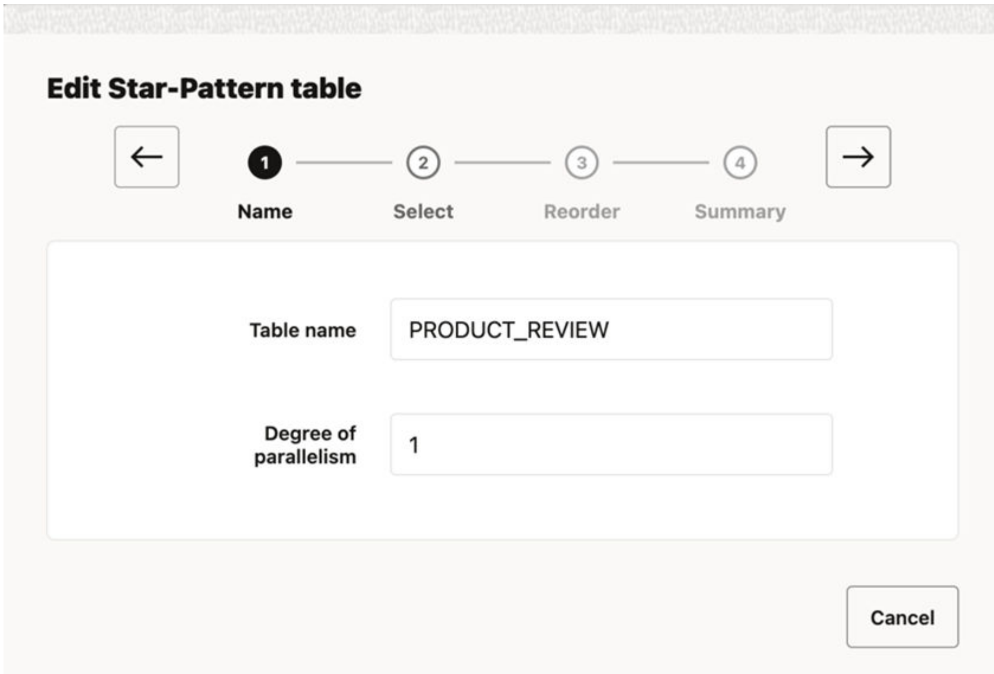


You can then choose to perform any of the following actions:

- Click the edit action icon if you wish to edit a result table.

  This initiates the edit result table workflow which is similar to the workflow for creating a result table. You can add or remove any property, edit the order of the properties, or change the settings for value columns or inverse property path.

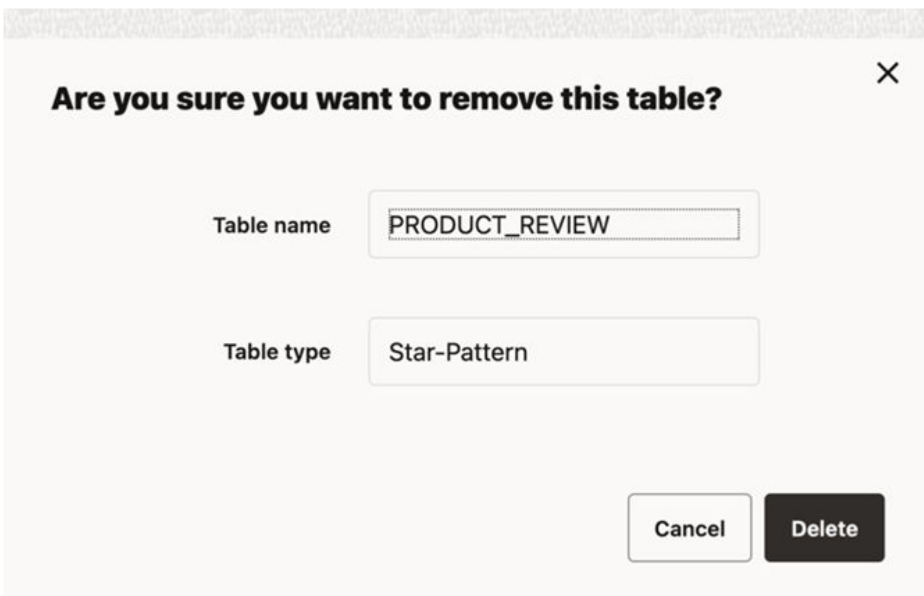
Figure 14-53 Edit Result Table



- Click the delete action icon if you wish to delete a result table.

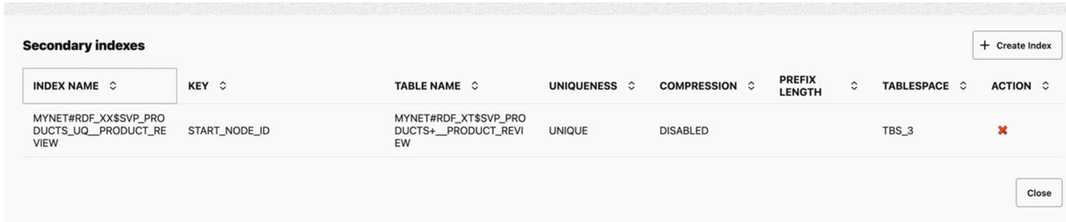
  A delete confirmation dialog is displayed as shown:

Figure 14-54 Deleting a Result table



- Click the view indexes action icon to view the secondary indexes for a result table as shown:

Figure 14-55 Viewing Secondary Indexes



You can perform the following actions on this page:

- Delete a specific index.
- Create a new index. See Creating an Index on a Result Table for more information.

- Creating an Index on a Result Table

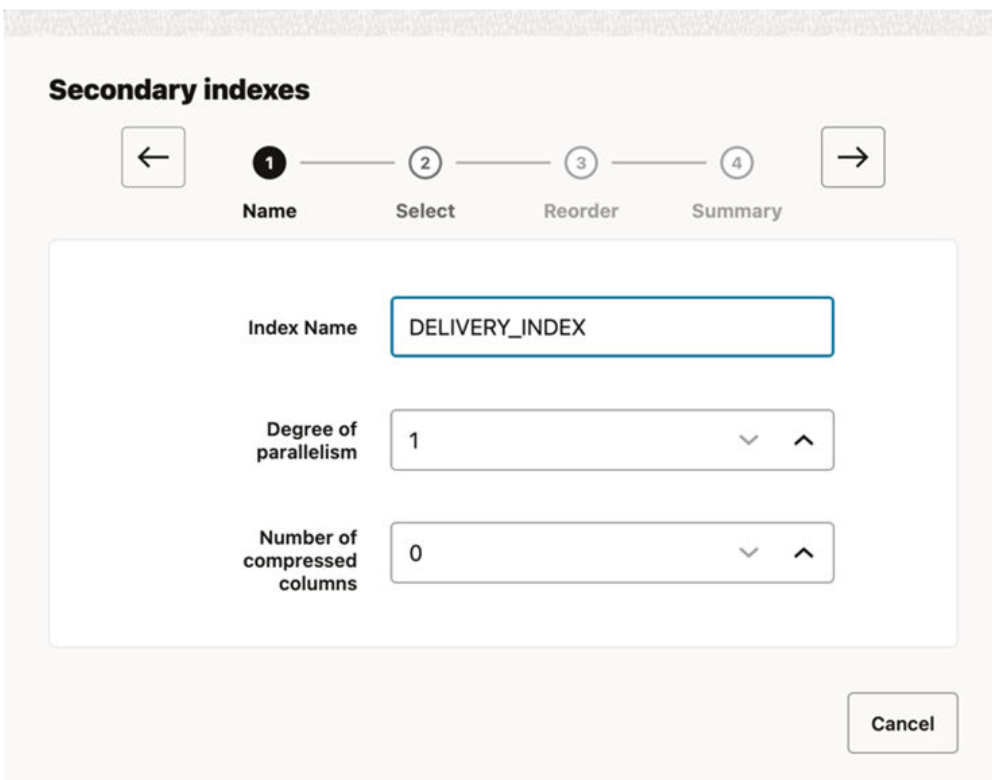
14.3.2.10.2.1 Creating an Index on a Result Table

You can create an index on a result table by performing the following steps:

1. Click **+ Create Index** on the top right corner of the **Secondary Indexes** page.

   The **Secondary indexes** wizard opens and displays the first (**Name**) step of the workflow as shown:

   Figure 14-56 Step1: Defining the Index Name

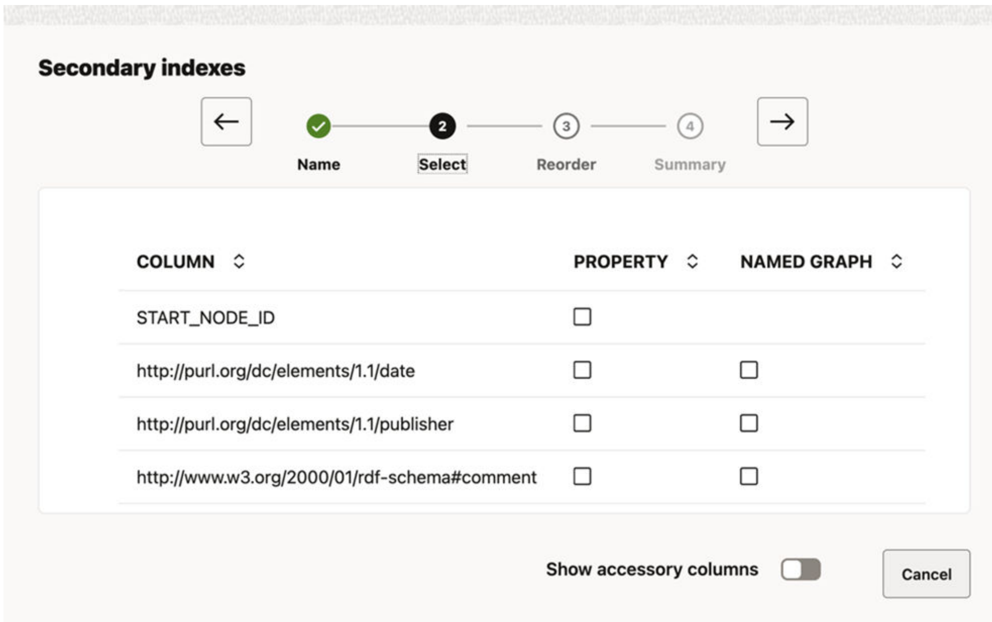
   

2. Enter the **Index Name**.

3. Optionally, select the **Degree of parallelism** and the **Number of compressed columns**.
4. Click → to move to the next step.

   The second (**Select**) step of the workflow opens as shown:

Figure 14-57 Step2: Selecting the Properties

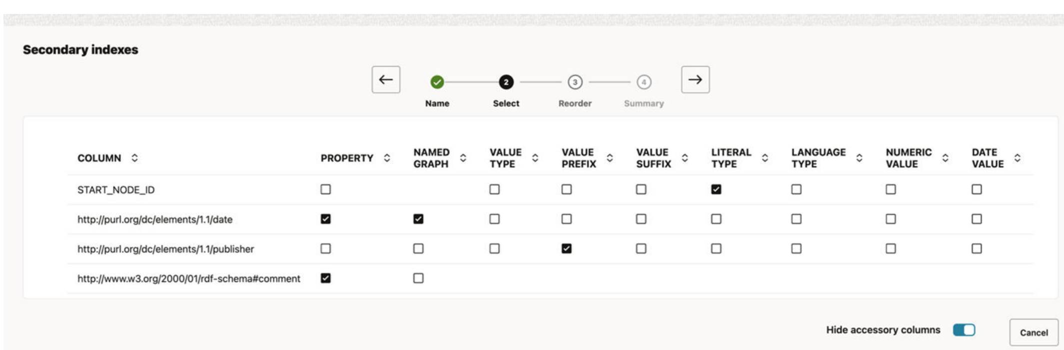


5. Select the required properties, graph values, or accessory columns.

   In contrast to previous versions, 23ai allows you to include any accessory column or the graph value without the need to include the property itself. This gives you complete flexibility and control when creating an index for a result table.

   If any one of the properties in the table have accessory columns, then you can optionally switch ON the **Show accessory columns** to display all the columns as shown:

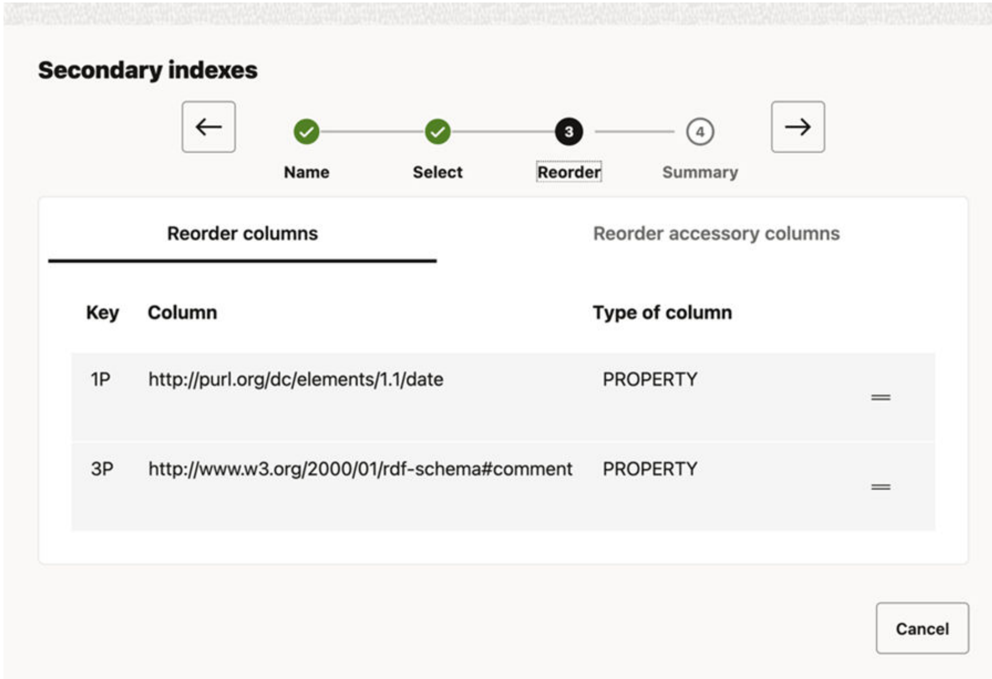
Figure 14-58 Step2: Selecting Accessory Columns



6. Click → to move to the next step.

   The third (**Reorder**) step of the workflow opens as shown:

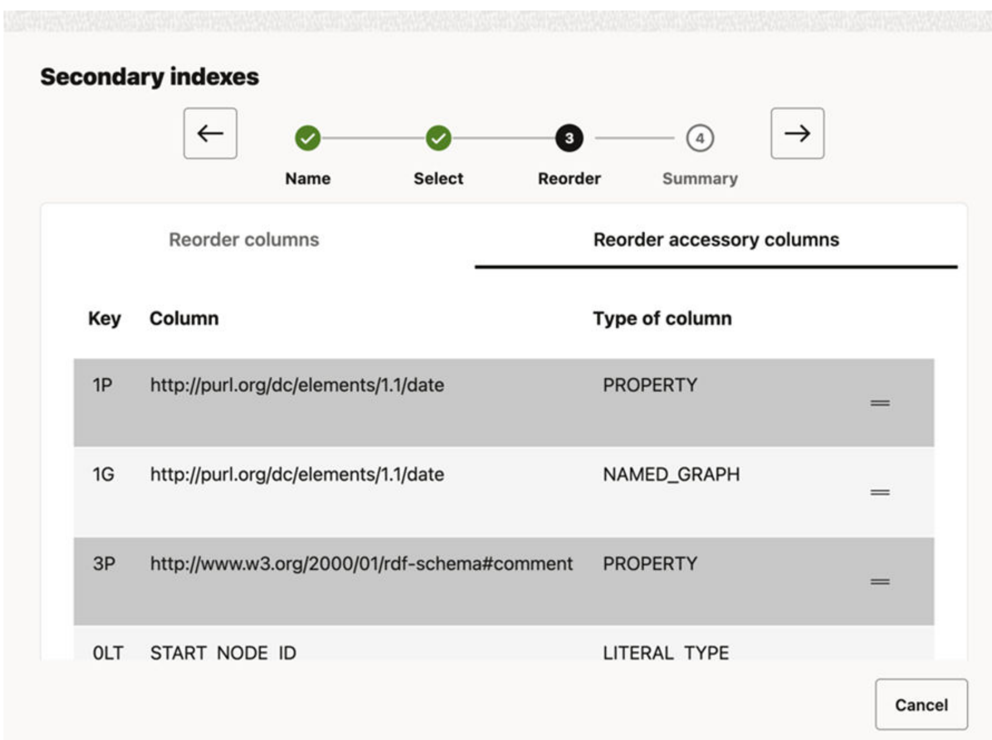
Figure 14-59 Step3: Reordering Properties



7. Drag and drop each property to reorder the selected columns as desired.

   If you added accessory columns in the preceding step, then in addition to the properties, you can also reorder the accessory columns as shown:

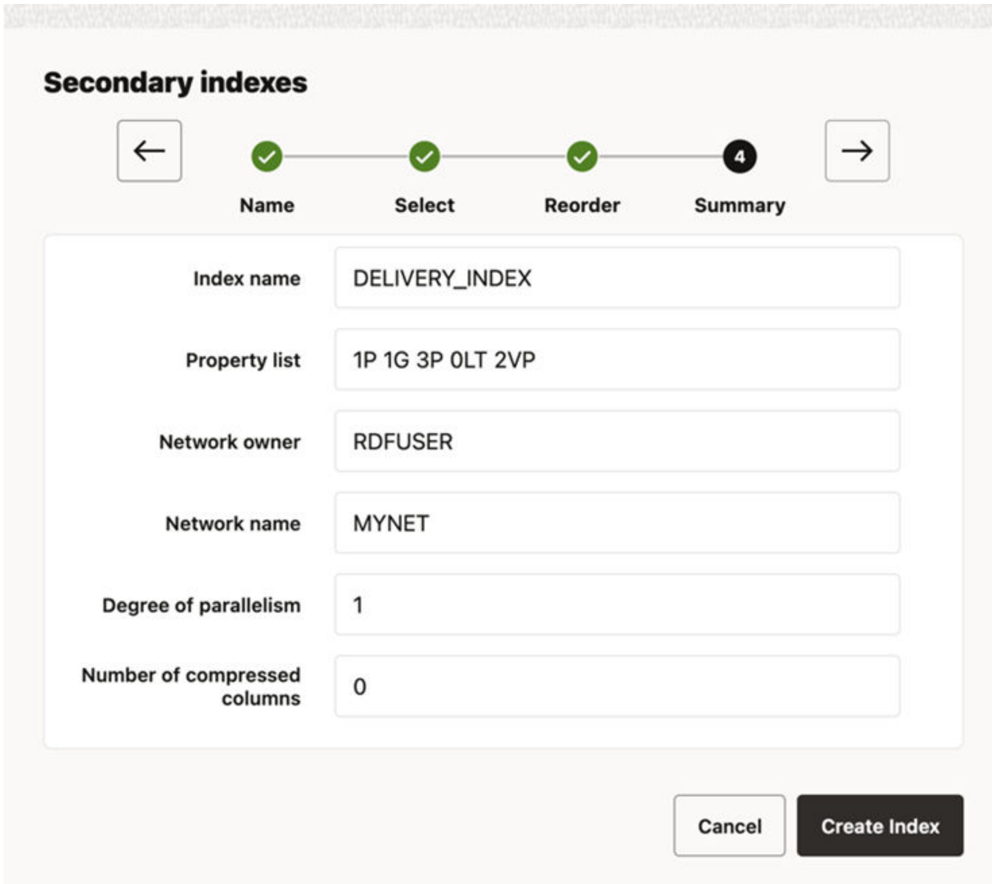
Figure 14-60 Step3: Reordering All the Columns



8. Click → to move to the next step.

   The final (**Summary**) step of the workflow opens as shown:

   Figure 14-61 Step4: Reviewing the Selections

9. Review the summary and click **Create Index** to create the new index for the result table.

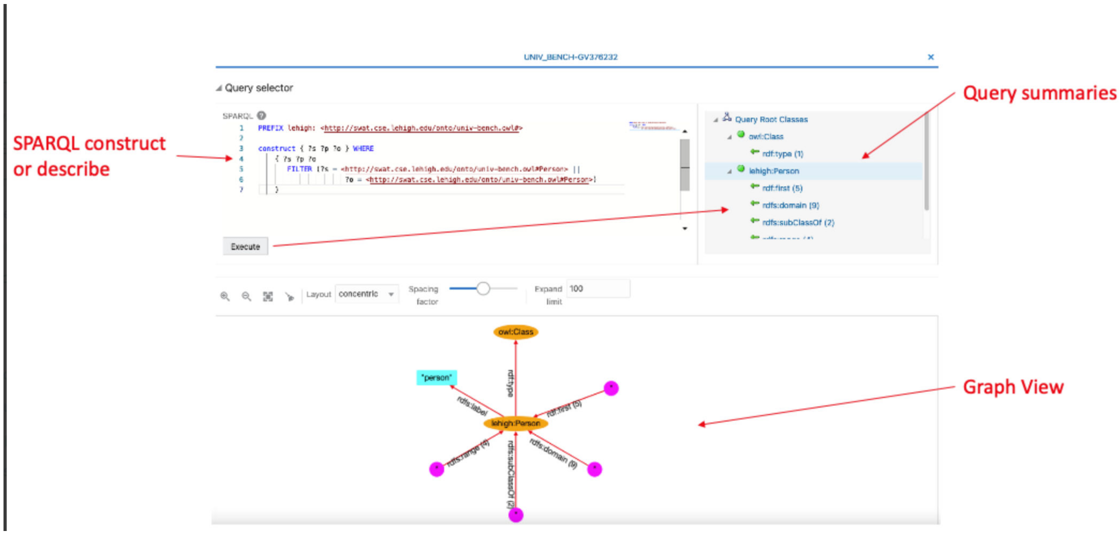
14.3.2.11 Advanced Graph View

The RDF Graph Query UI supports an advanced graph view feature that allows users to interact directly with the graph visualization. This is unlike the graph displayed on the RDF model editor or public component where the graph view is just an output of the SPARQL results on the paging table.

This section describes the advanced graph view component, starting from the execution of a SPARQL `CONSTRUCT` or SPARQL `DESCRIBE` query to advanced interaction with the graph visualization.

The main user interface (UI) elements of the advanced graph view component are as shown:

Figure 14-62 Advanced Graph View Components

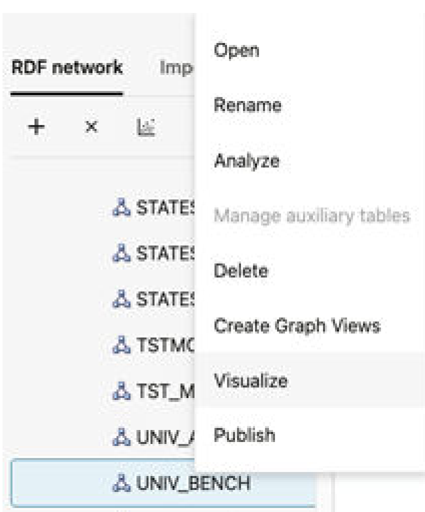


The following describes the UI components seen in the preceding figure:

- SPARQL **Query selector** contains:
  - A text area with the SPARQL query (must be SPARQL `CONSTRUCT` or SPARQL `DESCRIBE`)
  - A tree with the root classes summaries (counts of incoming and outgoing predicates) resulting from the SPARQL query
- A graph view area that displays the graph with the RDF nodes and edges

To access the advanced graph view feature, right-click on the RDF model and select **Visualize** as shown:

Figure 14-63 Visualize Menu



- Query Selector Panel

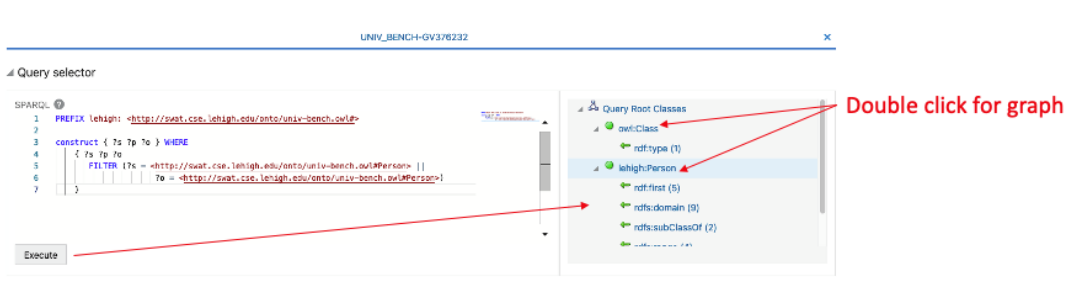
- Graph View

14.3.2.11.1 Query Selector Panel

To start using the advanced graph view feature, you must first execute a SPARQL `CONSTRUCT` or SPARQL `DESCRIBE` query. The resulting query output is organized as summaries (counts for incoming and outgoing predicates) for the root classes (in general URI or blank node values).

The following figure shows a SPARQL `CONSTRUCT` query that produces two root classes, `owl:Class` and `lehigh:Person`:

Figure 14-64 Query Selector



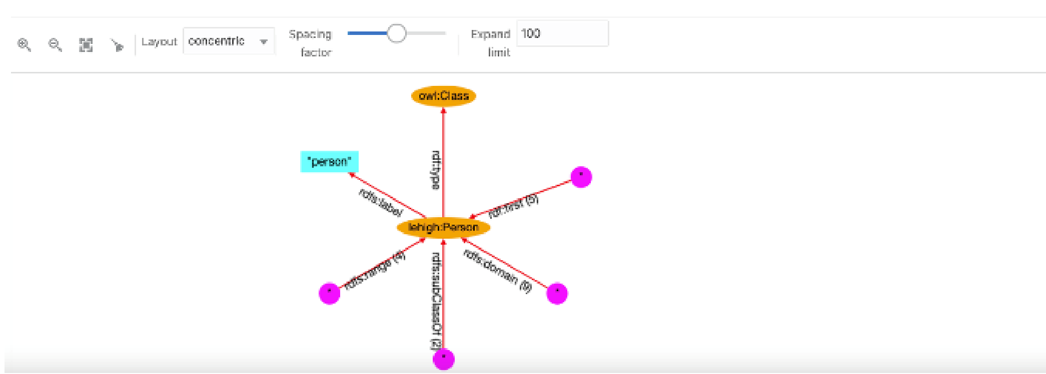
Each root class has its own summary of incoming and outgoing predicates. You can double click on a root class to view the graph representation in the graph view panel.

It is highly recommended to define `PREFIX` expressions on the query to shorten the result labels in the graph nodes. It also helps to consume less space for the graphic representation of the nodes. Some well known RDF SPARQL prefixes (such as *rdf*, *rdfs*, *owl*, and others) are automatically recognized and can be avoided in the query expression.

As seen in the preceding figure, you can double click the tree node to open the element as a graph in the graph view. You can then interact directly with the graph in the graph view without using the root tree nodes in the **Query selector** panel. This panel can be collapsed to provide more space on the page for the graph view.

The following figure shows the `owl:Class` and `lehigh:Person` elements displayed in the graph view.

Figure 14-65 Advanced Graph View



Note that in some cases the SPARQL query execution may generate several root classes. However, it is not necessary to add all the root classes to a graph. This also helps to maintain a clean and readable graph area.

14.3.2.11.2 Graph View

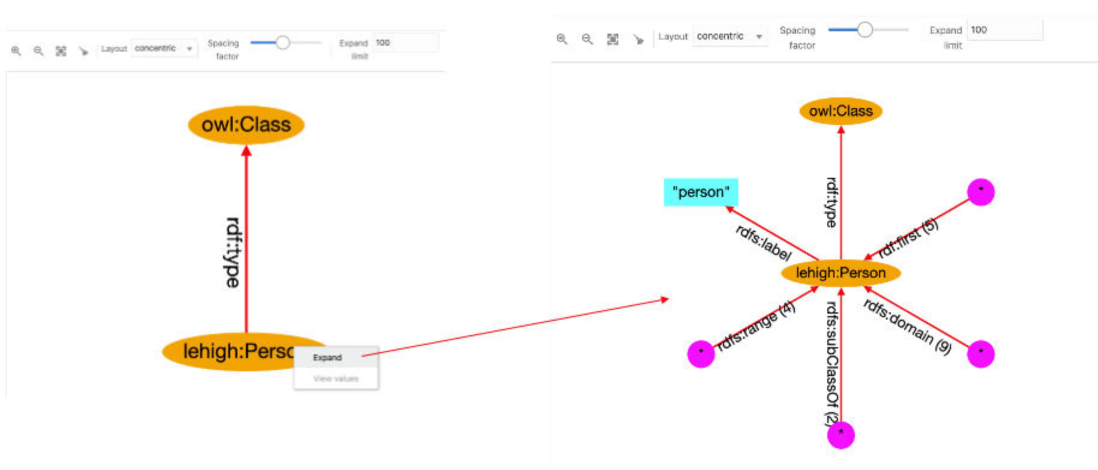
The graph view panel, where the graph is displayed, consists of the following components:

- A toolbar with the following options:
  - **Zoom Options:** Includes zoom in, zoom out, fit all, and clear all actions. Additionally, zoom in and out actions can be achieved with the mouse wheel. Drag to pan graph is also available.
  - **Layout:** A few built-in layouts (such as random, grid, circle, concentric, breadth first, and cose).
  - **Spacing factor:** A slider to adjust the spacing between nodes (useful for lengthy edges).
  - **Expand limit:** The maximum number of node entries that can be expanded for an edge.
- A drawing area with the RDF nodes and edges.

You can interact with the edges and nodes of the graph displayed in the graph view area. Initially, the graph displayed is based on the root class summaries (counts), but you can always expand the elements.

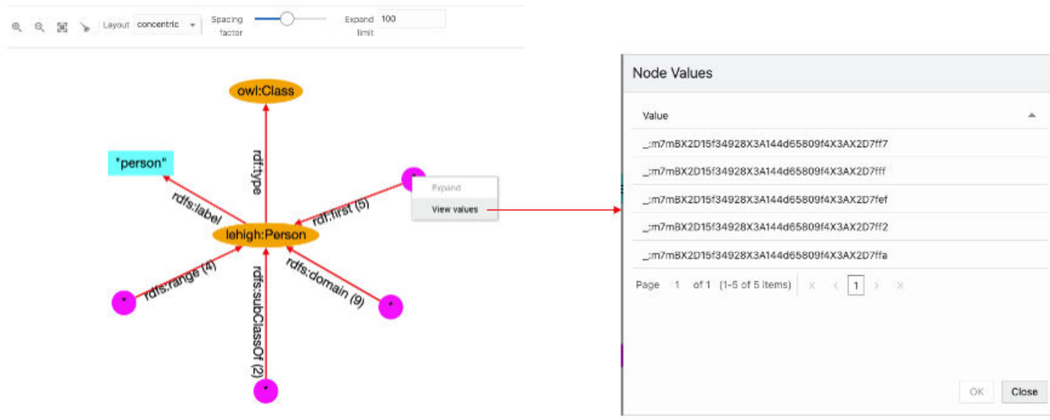
To expand a node in the graph, click on the node and then select **Expand**. New node elements with new edges linked to the selected node gets added to the graph. For example, in the following figure, the node *lehigh:Person* is shown expanded:

Figure 14-66 Expanding a Node



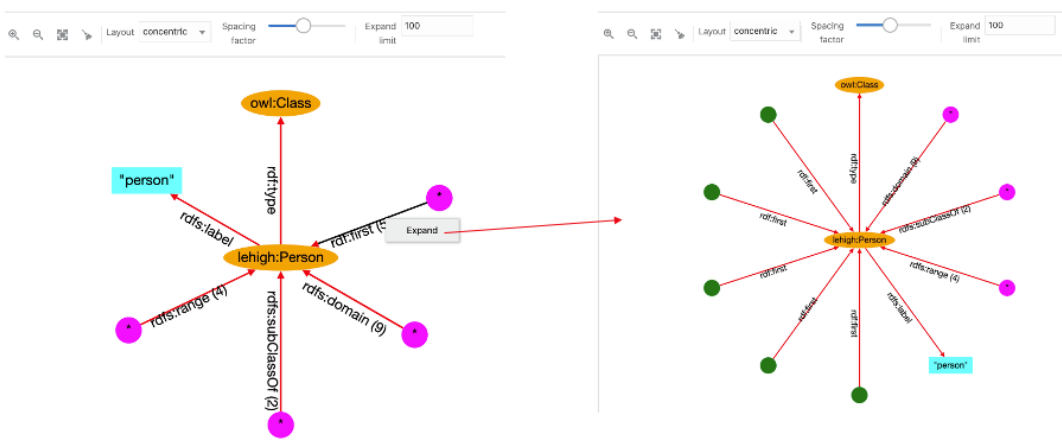
Star nodes (magenta color) contain the values associated with the edge predicate. To see these values, click on the node and select **View Values**:

Figure 14-67 Viewing Node Values



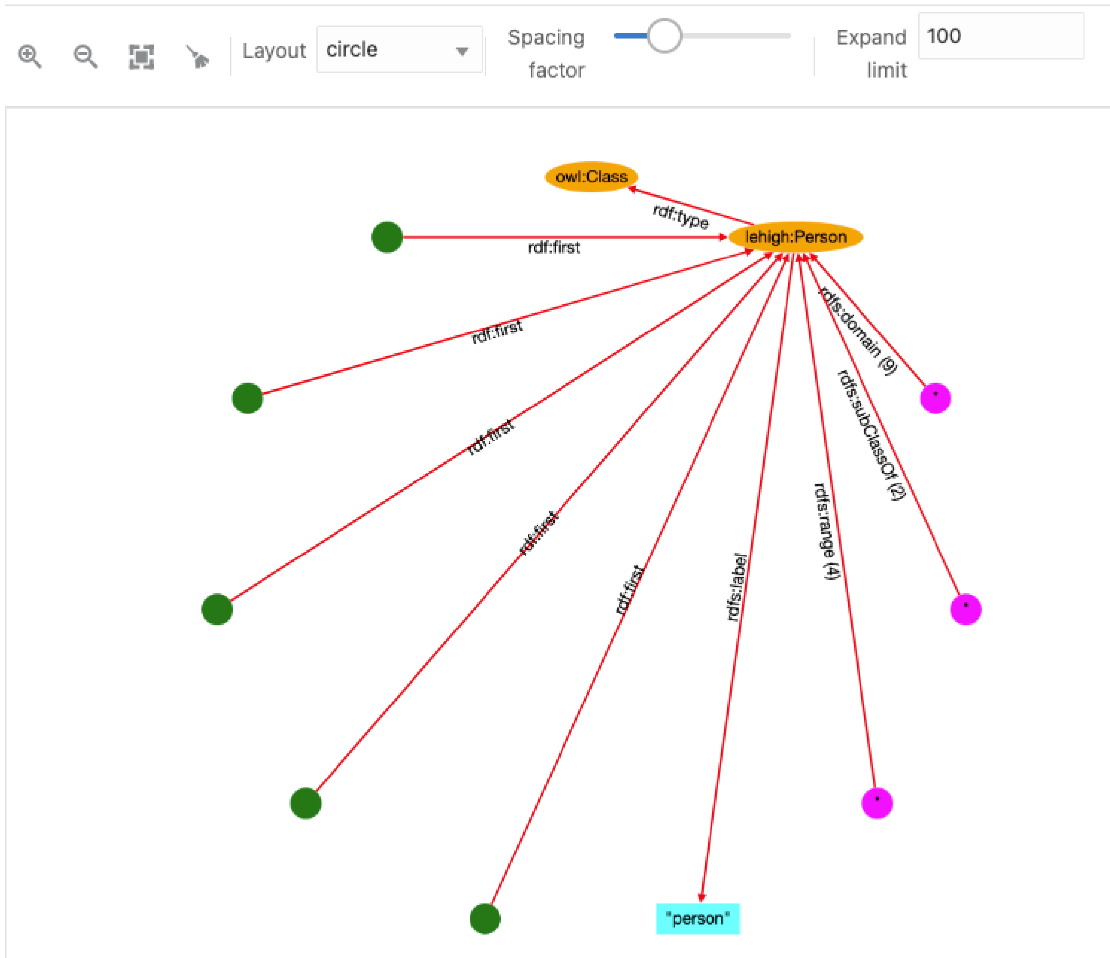
To expand the edge predicate summary, click on the edge and select **Expand**. Then the star node associated with it will be divided into new nodes and edges in the graph. However, if the expand limit value is lower than the summary count, then all the nodes will not be expanded. For example:

Figure 14-68 Expanding an Edge Predicate



The following figure displays the output for a circular layout:

Figure 14-69 Circular Layout Graph



The following basic conventions apply to the graph displayed in the graph view:

- URI nodes are displayed in orange color with labels inside the ellipse shape.
- Blank nodes are displayed in green color with circle shape. Mousing over the blank node shows its label value.
- Collapsed edges have the predicate with the count (if more than 1).
- Star nodes in magenta color and circle shape contain the values associated with the collapsed edge.
- Literal nodes are displayed with different colors depending on its type. A string literal is shown in cyan color with the label value. For long string values, the label length is reduced and mousing over literal node shows the full label value. Literals with datatype are displayed in different colors, and mousing over them shows the datatype name.

14.3.2.12 Database Views from RDF Models

You can create relational views from RDF models. These views can represent a vertex or an edge view of a graph.

SPARQL query patterns can be used as a declarative language for specifying how to build vertex and edge views from RDF data.

It is important to note the following when creating the vertex and edge views from an RDF model:

- The RDF model must have classes defined and the application uses a SPARQL query to retrieve the distinct classes defined on an RDF model. For example:

```
SELECT DISTINCT ?o
WHERE { ?s a ?o } order by ?o
```

- One or more RDF classes can define a vertex view. A vertex view consists of:
  - Database vertex view name
  - Key attribute name
  - Vertex properties from RDF class
- One or two vertex views can define an edge view. An edge view consists of:
  - Database edge view name
  - Source and destination vertex keys
  - Label property from RDF classes

The following sections explain the steps to create a database graph view:

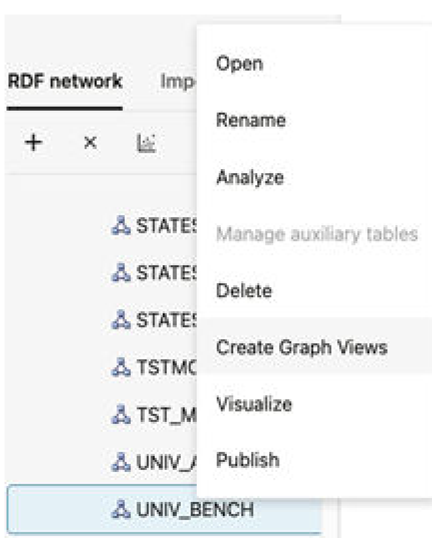
- Creating a Graph View
- Creating a Vertex View
- Creating an Edge View

14.3.2.12.1 Creating a Graph View

Perform the following steps to create a database graph view:

1. Right-click the RDF model to open the context menu as shown:

Figure 14-70 Create Graph View Option



2. Click **Create Graph Views**.

   The application opens an editor with the available RDF classes populated from a SPARQL query as shown:

Figure 14-71 RDF Classes

Note that the database graph views cannot be created if there are no RDF classes.

3. Add **Vertex Views** as required.

   See Creating a Vertex View for more information.

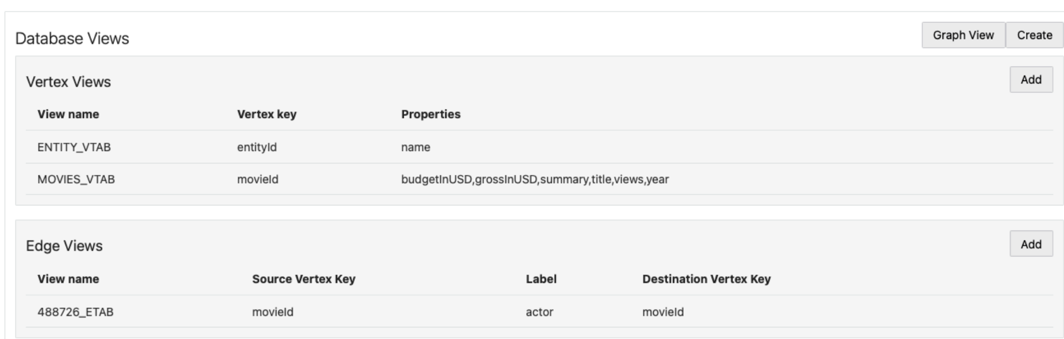
4. Add **Edge Views** as required.

   See Creating an Edge View for more information.

5. Review and verify the graph representation of the **Database Views**.

   The following figure shows a sample graph representation:

Figure 14-72 Sample Graph Definition



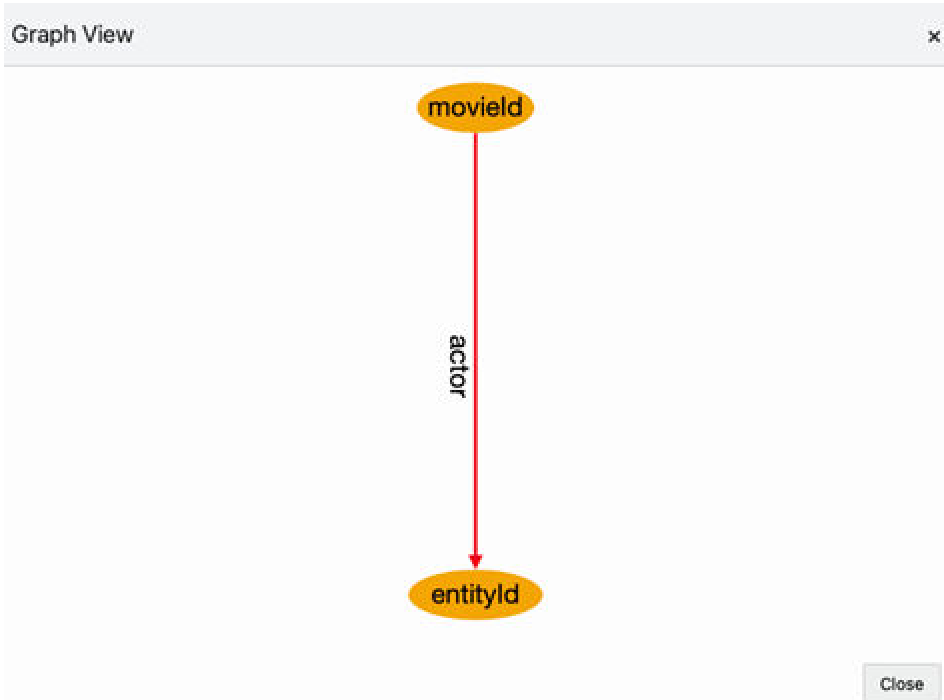
6. Optionally, you can hover over a table row and click the action menu icon to **Remove**, **Edit**, or **Preview** a specific vertex or an edge view.

Figure 14-73 Action Menu Options

7. Click **Graph View** to visualize the sample graph.

   Note that in a graph view, each node represents a vertex view and the link between nodes have an edge label. The following figure shows a sample graph visualization containing two vertex views with key attributes `movieId` and `entityId` which are linked by the `actor` edge label.

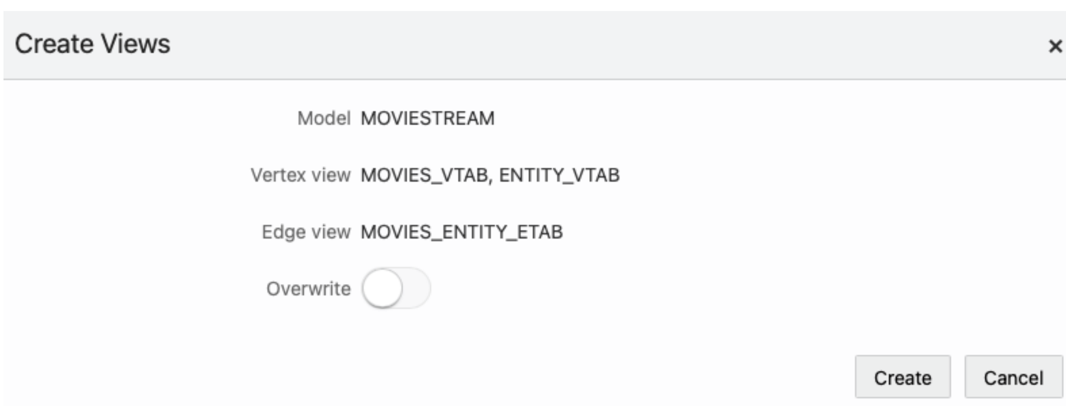
Figure 14-74 Graph Visualization for RDF Database Views



8. Click **Create** to create the RDF graph view in the database.

   The **Create Views** dialog opens as shown:

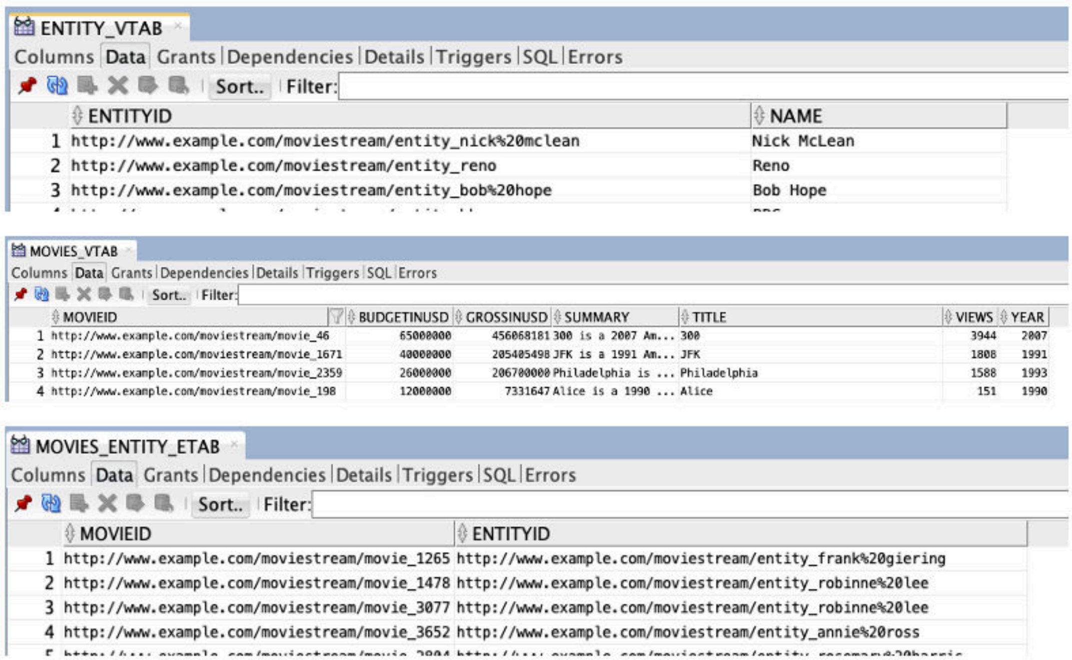
Figure 14-75 Create Views



   a. Optionally, switch **ON** the **Overwrite** option to replace any existing view definition.
   b. Click **Create**.

   The database graph view gets created.
   The following figure shows the views that are created in the database for the sample graph definition shown in step-5:

Figure 14-76   RDF Database Graph Views

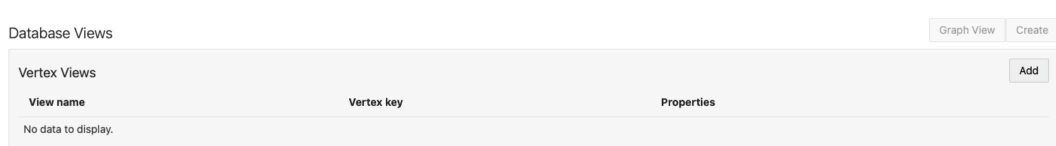


14.3.2.12.2 Creating a Vertex View

Perform the following steps to create a vertex view:

1. Click **Add** in the **Vertex Views** panel shown in the following figure:

Figure 14-77   Creating a Vertex View



2. Configure the **Vertex View Definition**.

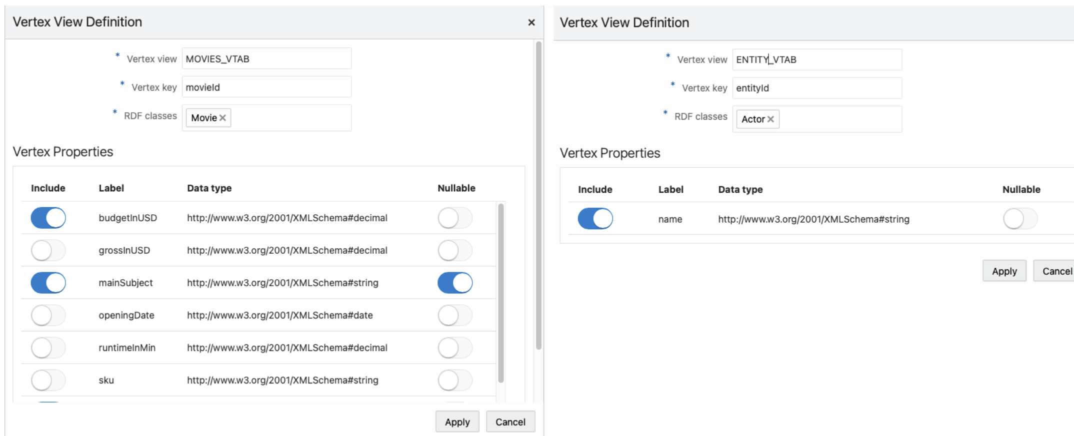
   Provide the following parameter values to define the vertex view:

   - **Vertex view:** Name of the vertex view. This will be used for querying the vertex.
   - **Vertex key:** Vertex key attribute.
   - **RDF classes:** One or more RDF classes. When RDF classes are added, the application retrieves the available properties for the class and lists them in the dialog. You can choose the properties to be added to the view. The **Vertex Properties** table has the following columns:
     - **Include:** At least one property must be included
     - **Label:** Property label
     - **Data type:** Displays the property data type
     - **Nullable:** At least one `FALSE` property must be included
       * `TRUE`: Vertices with `NULL` (missing) values for the property will be included.

* `FALSE`: Vertices with `NULL` (missing) values for the property will be excluded.

The following figure shows two examples of vertex view definitions (`movie` and `actor` entities):

Figure 14-78 Vertex View Definitions

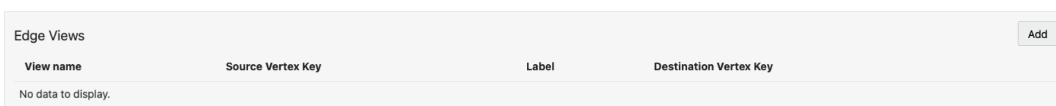


14.3.2.12.3 Creating an Edge View

An edge view can be defined using one or two vertex views.
To create an edge view:

1. Click **Add** in the **Edge Views** panel shown in the following figure:

   Figure 14-79 Edge Views

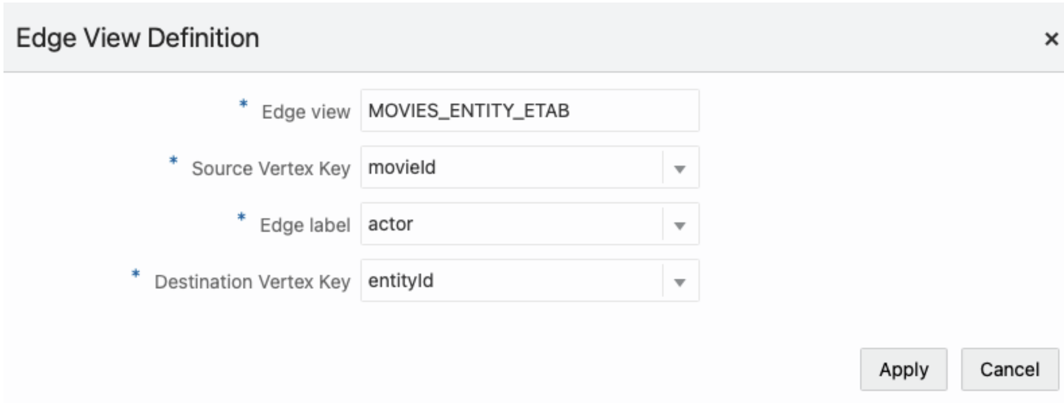
2. Configure the **Edge View Definition**.

   Provide the following parameter values to define the edge view:

   - **Edge view:** Name of the edge view. This will be used for querying the edge.
   - **Source Vertex key:** Source vertex key attribute.
   - **Edge label:** Edge label value.
   - **Destination Vertex key:** Destination vertex key attribute.

   In the following figure, the edge links the `movie` and `actor` entities:

Figure 14-80 Edge View Definition



14.3.3 Configuration Files for RDF Server and Client

The Graph Query UI application settings are determined by the JSON files that are included in the RDF Server and Client installation.

- `datasource.json`: File with RDF data source definitions.
- `general.json`: General configuration parameters.
- `proxy.json`: Proxy server parameters.
- `logging.json`: Logging settings.
- `seed.json`: Master seed key value generated at first deployment of the application. This is a unique value to be used for encrypting and decrypting passwords for Oracle data sources defined with credentials. This is an important file, and losing it will not allow you to encrypt or to decrypt passwords values.

On the server side, the directory `WEB-INF/workspace` is the default directory to store configuration information, logs, and temporary files. The configuration files are stored by default in `WEB-INF/workspace/config`.

> **Note:**
>
> If the RDF Graph Query application is deployed from an unexploded `.war` file, and if no JVM parameter is defined for the workspace folder location, then the default workspace location for the application is `WEB-INF/workspace`. However, any updates to the configuration, log, and temp files done by the application may be lost if the application is redeployed. Also, wallet data source files and published dataset files can be lost.
>
> To overcome this, you must start the application server, such as Weblogic or Tomcat, with the JVM parameter `oracle.rdf.workspace.dir` set. For example: `=Doracle.rdf.workspace.dir=/rdf/server/workspace`. The workspace folder must exist on the file system. Otherwise, the workspace folder defaults to `WEB-INF/workspace`.
>
> It is recommended to have a backup of the workspace folder, in case of redeploying the application on a different location. Copying the workspace folder contents to the location of the JVM parameter, allows to restore all configurations in new deployment.

- Data Sources JSON Configuration File
- General JSON configuration file
- Proxy JSON Configuration File
- Logging JSON Configuration File

14.3.3.1 Data Sources JSON Configuration File

The JSON file for data sources stores the general attributes of a data source, including specific properties associated with data source.

The following example shows a data source JSON file with two data sources: one an Oracle container data source defined on the application server, and the other an external data source.

```
{
  "datasources" : [
   {
    "name" : "rdfuser193c",
    "type" : "DATABASE",
    "description" : "19.3 Oracle database",
    "properties" : {
      "jndiName" : "jdbc/RDFUSER193c"
    }
   },
   {
     "name" : "dbpedia",
     "type" : "ENDPOINT",
     "description" : "Dbpedia RDF data - Dbpedia.org",
      "properties" : {
         "baseUrl" : "http://dbpedia.org/sparql",
         "provider" : "Dbpedia"
       }
    }
  ]
}
```

14.3.3.2 General JSON configuration file

The general JSON configuration file stores information related to SPARQL queries, JBDC parameters and upload parameters.

The JSON file includes the following parameters:

- **Maximum SPARQL rows:** Defines the limit of rows to be fetched for a SPARQL query. If a query returns more than this limit, the fetching process is stopped.
- **SPARQL Query Timeout:** Defines the time in seconds to wait for a query to complete.
- **Allow publishing:** Flag to enable public data source selection for using with SPARQL query endpoints.
- **Publishing data source:** The RDF data source to publish datasets.
- **JDBC Fetch size:** The fetch size parameter for JDBC queries.
- **JDBC CLOB Prefetch size:** Number of characters to be prefetched when retrieving large object values.
- **JDBC Batch size:** The batch parameter for JDBC updates.
- **Maximum file size to upload:** The maximum file size to be uploaded into server.

- **Maximum unzipped item size:** The maximum size for an item in a zip file.
- **Maximum unzipped total size:** The size limit for all entries in a zip file.
- **Maximum zip inflate multiplier:** Maximum allowed multiplier when inflating files.

These parameters can be updated as shown in the following figures

Figure 14-81 General SPARQL Parameters

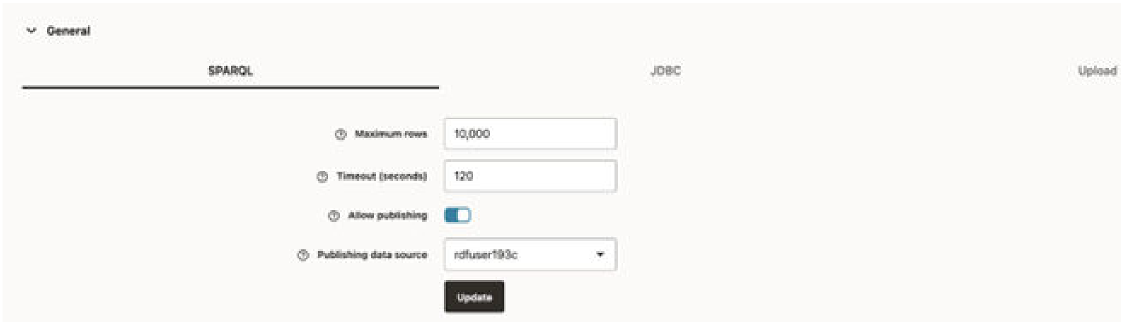


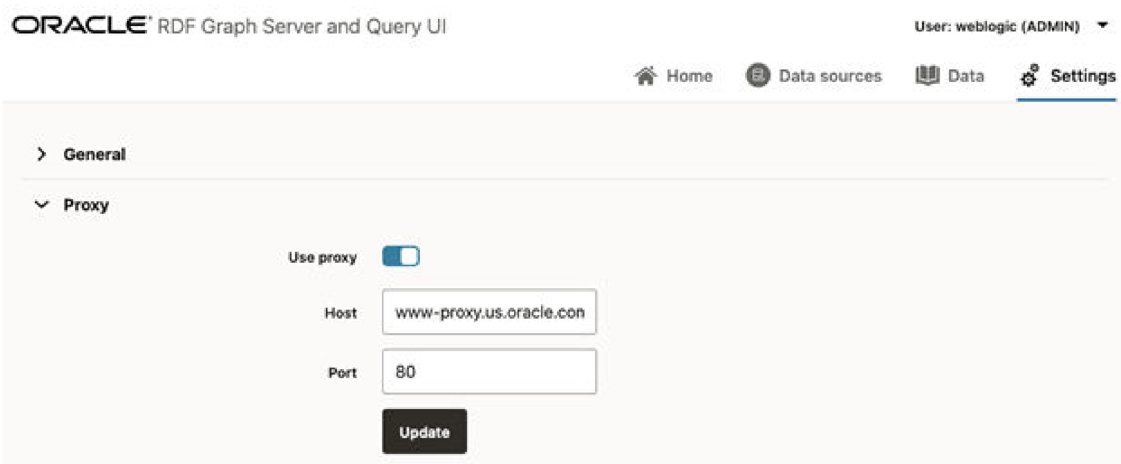
Figure 14-82 General JDBC Parameters

Figure 14-83 General File Upload Parameters

14.3.3.3 Proxy JSON Configuration File

The Proxy JSON configuration file contains proxy information for your enterprise network.

Figure 14-84 Proxy JSON Configuration File



The file includes the following parameters:

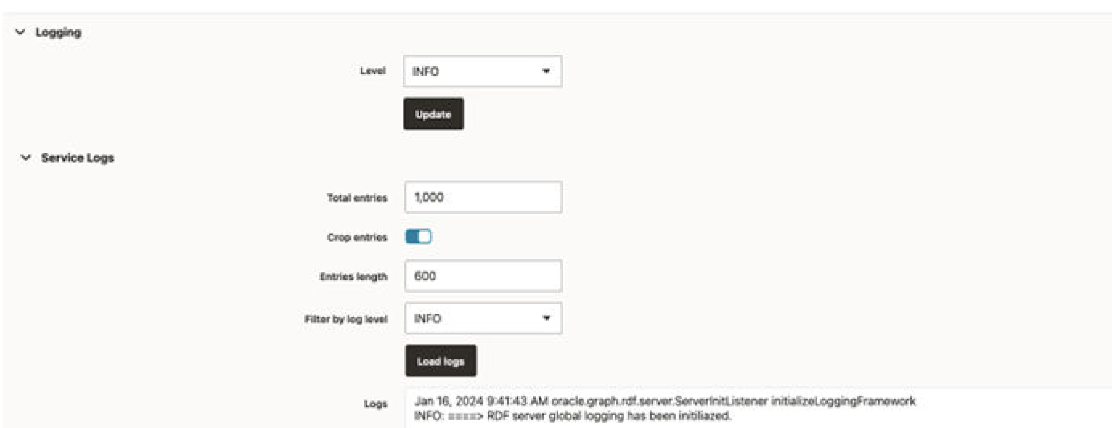
- **Use proxy:** flag to define if proxy parameters should be used.
- **Host:** proxy host value.
- **Port:** proxy port value.

14.3.3.4 Logging JSON Configuration File

The Logging JSON configuration file contains the logging settings. You can specify the logging level.

For Administrators and RDF users, it is also possible to load the logs for further analysis.

Figure 14-85 Logging JSON Configuration File



14.4 Accessibility

You can turned on or off the accessibility during the user session.

To disable accessibility, click on the username drop-down menu on the top right of the page, and turn off the **Accessibility** option.

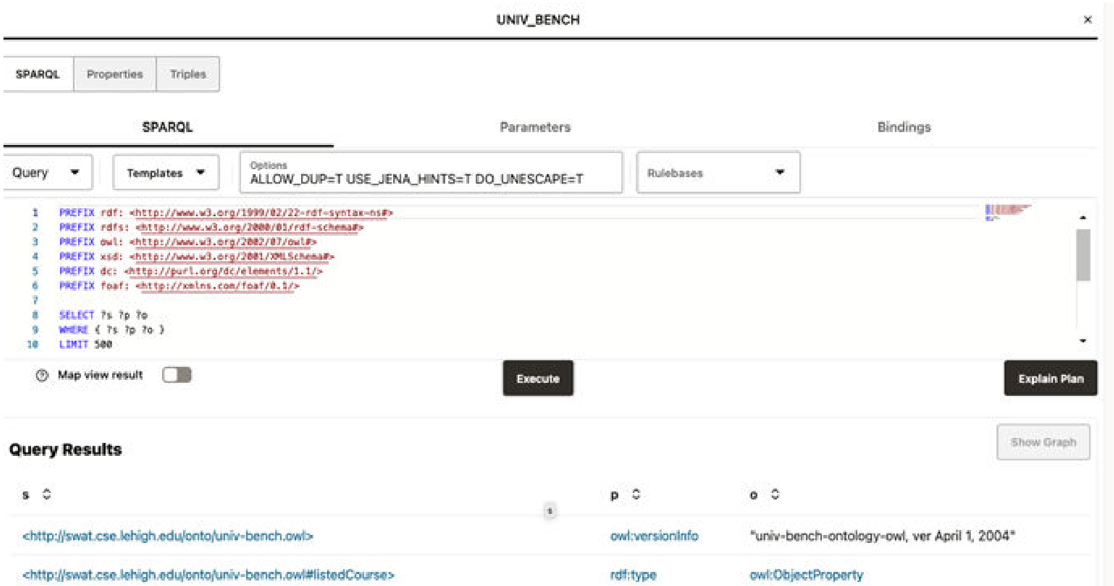
Figure 14-86 Disabled Accessibility

To enable accessibility, click on the username drop-down menu on the top right of the page, and turn on the **Accessibility** option.

Figure 14-87 Enabled Accessibility

When accessibility is turned on, the graph view of SPARQL queries is disabled.

Figure 14-88 Disabled Graph View



Part III
Reference Information

Part III provides reference information about RDF Semantic Graph subprograms.

This part contains the following chapters with reference information. To understand the examples in the reference chapters, you must understand the conceptual and data type information in RDF Semantic Graph Overview and OWL Concepts.

- SEM_APIS Package Subprograms
  The SEM_APIS package contains subprograms (functions and procedures) for working with the Resource Description Framework (RDF) and Web Ontology Language (OWL) in an Oracle database.
- SEM_PERF Package Subprograms
  The SEM_PERF package contains subprograms for examining and enhancing the performance of the Resource Description Framework (RDF) and Web Ontology Language (OWL) support in an Oracle database.
- SEM_RDFCTX Package Subprograms
  The SEM_RDFCTX package contains subprograms (functions and procedures) to manage extractor policies and semantic indexes created for documents.
- SEM_RDFSA Package Subprograms
  The SEM_RDFSA package contains subprograms (functions and procedures) for providing fine-grained access control to RDF data using Oracle Label Security (OLS).

15 SEM_APIS Package Subprograms

The SEM_APIS package contains subprograms (functions and procedures) for working with the Resource Description Framework (RDF) and Web Ontology Language (OWL) in an Oracle database.

To use the subprograms in this chapter, you must understand the conceptual and usage information in RDF Graph Overview and OWL Concepts

> **Note:**
>
> If you are using an Autonomous Database serverless instance, then all the subprograms related to the following features require Oracle JVM to be enabled:
>
> - Querying RDF data using SEM_APIS.SPARQL_TO_SQL
> - Querying RDF data using the SEM_MATCH table function
> - Performing SPARQL update operations on an RDF graph
> - Using the RDF graph adapter for Eclipse RDF4J
> - User-defined inferencing and querying
>
> See Use Oracle Java in *Using Oracle Autonomous Database Serverless* to enable Oracle JVM on your Autonomous Database instance.

This chapter provides reference information about the subprograms, listed in alphabetical order.

- SEM_APIS.ADD_DATATYPE_INDEX
- SEM_APIS.ADD_NETWORK_INDEX
- SEM_APIS.ADD_SEM_INDEX
- SEM_APIS.ALTER_DATATYPE_INDEX
- SEM_APIS.ALTER_ENTAILMENT
- SEM_APIS.ALTER_INDEX_ON_INFERRED_GRAPH
- SEM_APIS.ALTER_INDEX_ON_RDF_GRAPH
- SEM_APIS.ALTER_INFERRED_GRAPH
- SEM_APIS.ALTER_MODEL
- SEM_APIS.ALTER_RDF_GRAPH
- SEM_APIS.ALTER_RDF_INDEXES
- SEM_APIS.ALTER_RESULT_TAB
- SEM_APIS.ALTER_SEM_INDEX_ON_ENTAILMENT
- SEM_APIS.ALTER_SEM_INDEX_ON_MODEL

- SEM_APIS.ALTER_SEM_INDEXES
- SEM_APIS.ALTER_SPM_TAB
- SEM_APIS.ANALYZE_ENTAILMENT
- SEM_APIS.ANALYZE_INFERRED_GRAPH
- SEM_APIS.ANALYZE_MODEL
- SEM_APIS.ANALYZE_RDF_GRAPH
- SEM_APIS.APPEND_RDF_NETWORK_DATA
- SEM_APIS.APPEND_SEM_NETWORK_DATA
- SEM_APIS.BUILD_RESULT_TAB
- SEM_APIS.BUILD_SPM_TAB
- SEM_APIS.BULK_LOAD_FROM_STAGING_TABLE
- SEM_APIS.BULK_LOAD_RDF_GRAPH
- SEM_APIS.CLEANUP_BNODES
- SEM_APIS.CLEANUP_FAILED
- SEM_APIS.COMPOSE_RDF_TERM
- SEM_APIS.CONVERT_TO_GML311_LITERAL
- SEM_APIS.CONVERT_TO_WKT_LITERAL
- SEM_APIS.CREATE_ENTAILMENT
- SEM_APIS.CREATE_INDEX_ON_RESULT_TAB
- SEM_APIS.CREATE_INDEX_ON_SPM_TAB
- SEM_APIS.CREATE_INFERRED_GRAPH
- SEM_APIS.CREATE_MATERIALIZED_VIEW
- SEM_APIS.SEM_APIS.CREATE_MV_BITMAP_INDEX
- SEM_APIS.CREATE_RDF_GRAPH
- SEM_APIS.CREATE_RDF_GRAPH_COLLECTION
- SEM_APIS.CREATE_RDF_NETWORK
- SEM_APIS.CREATE_RDFVIEW_GRAPH
- SEM_APIS.CREATE_RDFVIEW_MODEL
- SEM_APIS.CREATE_RULEBASE
- SEM_APIS.CREATE_SEM_MODEL
- SEM_APIS.CREATE_SEM_NETWORK
- SEM_APIS.CREATE_SEM_SQL
- SEM_APIS.CREATE_SOURCE_EXTERNAL_TABLE
- SEM_APIS.CREATE_SPARQL_UPDATE_TABLES
- SEM_APIS.CREATE_VIRTUAL_MODEL
- SEM_APIS.DELETE_ENTAILMENT_STATS
- SEM_APIS.DELETE_MODEL_STATS
- SEM_APIS.DISABLE_CHANGE_TRACKING

- SEM_APIS.DISABLE_INC_INFERENCE
- SEM_APIS.DISABLE_INMEMORY
- SEM_APIS.DISABLE_INMEMORY_FOR_ENT
- SEM_APIS.DISABLE_INMEMORY_FOR_MODEL
- SEM_APIS.DISABLE_INMEMORY_FOR_INF_GRAPH
- SEM_APIS.DISABLE_INMEMORY_FOR_RDF_GRAPH
- SEM_APIS.DISABLE_NETWORK_SHARING
- SEM_APIS.DROP_DATATYPE_INDEX
- SEM_APIS.DROP_ENTAILMENT
- SEM_APIS.DROP_INFERRED_GRAPH
- SEM_APIS.DROP_MATERIALIZED_VIEW
- SEM_APIS.DROP_MV_BITMAP_INDEX
- SEM_APIS.DROP_NETWORK_INDEX
- SEM_APIS.DROP_RDF_GRAPH
- SEM_APIS.DROP_RDF_GRAPH_COLLECTION
- SEM_APIS.DROP_RDF_NETWORK
- SEM_APIS.DROP_RDFVIEW_GRAPH
- SEM_APIS.DROP_RDFVIEW_MODEL
- SEM_APIS.DROP_RESULT_TAB
- SEM_APIS.DROP_RULEBASE
- SEM_APIS.DROP_SEM_INDEX
- SEM_APIS.DROP_SEM_MODEL
- SEM_APIS.DROP_SEM_NETWORK
- SEM_APIS.DROP_SEM_SQL
- SEM_APIS.DROP_SPARQL_UPDATE_TABLES
- SEM_APIS.DROP_SPM_TAB
- SEM_APIS.DROP_USER_INFERENCE_OBJS
- SEM_APIS.DROP_VIRTUAL_MODEL
- SEM_APIS.ENABLE_CHANGE_TRACKING
- SEM_APIS.ENABLE_INC_INFERENCE
- SEM_APIS.ENABLE_INMEMORY
- SEM_APIS.ENABLE_INMEMORY_FOR_ENT
- SEM_APIS.ENABLE_INMEMORY_FOR_INF_GRAPH
- SEM_APIS.ENABLE_INMEMORY_FOR_MODEL
- SEM_APIS.ENABLE_INMEMORY_FOR_RDF_GRAPH
- SEM_APIS.ENABLE_NETWORK_SHARING
- SEM_APIS.ESCAPE_CLOB_TERM
- SEM_APIS.ESCAPE_CLOB_VALUE

- SEM_APIS.ESCAPE_RDF_TERM
- SEM_APIS.ESCAPE_RDF_VALUE
- SEM_APIS.EXPORT_ENTAILMENT_STATS
- SEM_APIS.EXPORT_MODEL_STATS
- SEM_APIS.EXPORT_RDFVIEW_GRAPH
- SEM_APIS.EXPORT_RDFVIEW_MODEL
- SEM_APIS.GATHER_SPM_INFO
- SEM_APIS.GET_CHANGE_TRACKING_INFO
- SEM_APIS.GET_INC_INF_INFO
- SEM_APIS.GET_MODEL_ID
- SEM_APIS.GET_MODEL_NAME
- SEM_APIS.GET_PLAN_COST
- SEM_APIS.GET_SQL
- SEM_APIS.GET_TRIPLE_ID
- SEM_APIS.GETV$DATETIMETZVAL
- SEM_APIS.GETV$DATETZVAL
- SEM_APIS.GETV$GEOMETRYVAL
- SEM_APIS.GETV$NUMERICVAL
- SEM_APIS.GETV$STRINGVAL
- SEM_APIS.GETV$TIMETZVAL
- SEM_APIS.GRANT_MODEL_ACCESS_PRIV
- SEM_APIS.GRANT_MODEL_ACCESS_PRIVS
- SEM_APIS.GRANT_NETWORK_ACCESS_PRIVS
- SEM_APIS.GRANT_NETWORK_SHARING_PRIVS
- SEM_APIS.GRANT_RDF_GRAPH_ACCESS_PRIV
- SEM_APIS.GRANT_RDF_GRAPH_ACCESS_PRIVS
- SEM_APIS.IMPORT_ENTAILMENT_STATS
- SEM_APIS.IMPORT_MODEL_STATS
- SEM_APIS.IS_TRIPLE
- SEM_APIS.LOAD_INTO_STAGING_TABLE
- SEM_APIS.LOOKUP_ENTAILMENT
- SEM_APIS.MERGE_MODELS
- SEM_APIS.MERGE_RDF_GRAPHS
- SEM_APIS.MIGRATE_DATA_TO_CURRENT
- SEM_APIS.MIGRATE_DATA_TO_STORAGE_V2
- SEM_APIS.MOVE_RDF_NETWORK_DATA
- SEM_APIS.MOVE_SEM_NETWORK_DATA
- SEM_APIS.PURGE_UNUSED_VALUES

- SEM_APIS.REFRESH_MATERIALIZED_VIEW
- SEM_APIS.REFRESH_NETWORK_INDEX_INFO
- SEM_APIS.REFRESH_SEM_NETWORK_INDEX_INFO
- SEM_APIS.RENAME_ENTAILMENT
- SEM_APIS.RENAME_INFERRED_GRAPH
- SEM_APIS.RENAME_MODEL
- SEM_APIS.RENAME_RDF_GRAPH
- SEM_APIS.RES2VID
- SEM_APIS.RESTORE_RDF_NETWORK_DATA
- SEM_APIS.RESTORE_SEM_NETWORK_DATA
- SEM_APIS.REVOKE_MODEL_ACCESS_PRIV
- SEM_APIS.REVOKE_MODEL_ACCESS_PRIVS
- SEM_APIS.REVOKE_NETWORK_ACCESS_PRIVS
- SEM_APIS.REVOKE_NETWORK_SHARING_PRIVS
- SEM_APIS.REVOKE_RDF_GRAPH_ACCESS_PRIV
- SEM_APIS.REVOKE_RDF_GRAPH_ACCESS_PRIVS
- SEM_APIS.SEM_SQL_COMPILE
- SEM_APIS.SET_ENTAILMENT_STATS
- SEM_APIS.SET_MODEL_STATS
- SEM_APIS.SPARQL_TO_SQL
- SEM_APIS.SWAP_NAMES
- SEM_APIS.TRUNCATE_SEM_MODEL
- SEM_APIS.TRUNCATE_RDF_GRAPH
- SEM_APIS.UNESCAPE_CLOB_TERM
- SEM_APIS.UNESCAPE_CLOB_VALUE
- SEM_APIS.UNESCAPE_RDF_TERM
- SEM_APIS.UNESCAPE_RDF_VALUE
- SEM_APIS.UPDATE_MODEL
- SEM_APIS.UPDATE_RDF_GRAPH
- SEM_APIS.VALIDATE_ENTAILMENT
- SEM_APIS.VALIDATE_GEOMETRIES
- SEM_APIS.VALIDATE_INFERRED_GRAPH
- SEM_APIS.VALIDATE_MODEL
- SEM_APIS.VALIDATE_RDF_GRAPH
- SEM_APIS.VALUE_NAME_PREFIX
- SEM_APIS.VALUE_NAME_SUFFIX

15.1 SEM_APIS.ADD_DATATYPE_INDEX

Format

```
SEM_APIS.ADD_DATATYPE_INDEX(
     datatype        IN VARCHAR2,
     tablespace_name IN VARCHAR2 DEFAULT NULL,
     parallel        IN PLS_INTEGER DEFAULT NULL,
     online          IN BOOLEAN DEFAULT FALSE,
     options         IN VARCHAR2 DEFAULT NULL,
     network_owner   IN VARCHAR2 DEFAULT NULL,
     network_name    IN VARCHAR2 DEFAULT NULL);
```

Description

Adds a data type index for the specified data type to an RDF network.

Parameters

datatype
URI of the data type to index.

tablespace_name
Destination tablespace for the index.

parallel
Degree of parallelism to use when building the index.

online
TRUE allows DML operations affecting the index during creation of the index; FALSE (the default) does not allow DML operations affecting the index during creation of the index.

options
String specifying options for index creation using the form *OPTION_NAME=option_value*. Supported options associated with spatial index creation are SRID, TOLERANCE, and DIMENSIONS. For materialized spatial index creation, use MATERIALIZE=T. Supported options associated with text index creation are PREFIX_INDEX, PREFIX_MIN_LENGTH, PREFIX_MAX_LENGTH, SUBSTRING_INDEX and LOGGING. For function-based numeric or dateTime index creation, use FUNCTION=T. The option name keywords are case sensitive and must be specified in uppercase.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

You must have DBA privileges to call this procedure.

For more information about data type indexing, see Using Data Type Indexes.

For information about creating a like index, see the lightweight text search material in Full-Text Search.

For information about creating a data type index on RDF spatial data, see Indexing Spatial Data.

For information about RDF network types and options, see RDF Networks.

Examples

The following example creates an index on `xsd:string` typed literals and plain literals in the MY_TBS tablespace.

```
EXECUTE SEM_APIS.ADD_DATATYPE_INDEX('http://www.w3.org/2001/XMLSchema#string',
tablespace_name=>'MY_TBS', parallel=>4);
```

15.2 SEM_APIS.ADD_NETWORK_INDEX

Format

```
SEM_APIS.ADD_NETWORK_INDEX(
    index_code          IN VARCHAR2,
    tablespace_name     IN VARCHAR2 DEFAULT NULL,
    compression_length  IN NUMBER(38) DEFAULT NULL,
    options             IN VARCHAR2 DEFAULT NULL,
    network_owner       IN VARCHAR2 DEFAULT NULL,
    network_name        IN VARCHAR2 DEFAULT NULL);
```

Description

Creates an RDF network index that results in creation of a non-unique B-tree index in UNUSABLE status for each of the existing RDF graphs and inferred RDF graphs of the RDF network.

Parameters

index_code
Index code string.

tablespace_name
Destination tablespace for the index.

compression_length

options

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

You must have DBA privileges to call this procedure.

For an explanation of RDF network indexes, see Using RDF Network Indexes.

For information about RDF network types and options, see RDF Networks.

Examples

The following example creates an RDF network index with the index code string `CSPGM` on the RDF graphs and inferred RDF graphs of the RDF network.

```
EXECUTE SEM_APIS.ADD_NETWORK_INDEX('CSPGM');
```

15.3 SEM_APIS.ADD_SEM_INDEX

Format

```
SEM_APIS.ADD_SEM_INDEX(
    index_code          IN VARCHAR2,
    tablespace_name     IN VARCHAR2 DEFAULT NULL,
    compression_length  IN NUMBER(38) DEFAULT NULL,
    options             IN VARCHAR2 DEFAULT NULL,
    network_owner       IN VARCHAR2 DEFAULT NULL,
    network_name        IN VARCHAR2 DEFAULT NULL);
```

> **Note:**
>
> This subprogram will be deprecated in a future release. It is recommended that you use the SEM_APIS.ADD_NETWORK_INDEX subprogram instead.

Description

Creates a semantic network index that results in creation of a non-unique B-tree index in UNUSABLE status for each of the existing models and entailments of the semantic network.

Parameters

index_code
Index code string.

tablespace_name
Destination tablespace for the index.

compression_length

options

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

You must have DBA privileges to call this procedure.

For an explanation of semantic network indexes, see Using Semantic Network Indexes.

For information about semantic network types and options, see RDF Networks.

Examples

The following example creates a semantic network index with the index code string `CSPGM` on the models and entailments of the semantic network.

```
EXECUTE SEM_APIS.ADD_SEM_INDEX('CSPGM');
```

15.4 SEM_APIS.ALTER_DATATYPE_INDEX

Format

```
SEM_APIS.ALTER_DATATYPE_INDEX(
     datatype        IN VARCHAR2,
     command         IN VARCHAR2,
     tablespace_name IN VARCHAR2 DEFAULT NULL,
     parallel        IN PLS_INTEGER DEFAULT NULL,
     online          IN BOOLEAN DEFAULT FALSE,
      network_owner   IN VARCHAR2 DEFAULT NULL,
      network_name    IN VARCHAR2 DEFAULT NULL);
```

Description

Alters a data type index.

Parameters

datatype
URI of the data type to index.

command
String specifying the command to be performed: `REBUILD` to rebuild the data type index, or `UNUSABLE` to marks the data type index as unusable. The value for this parameter is not case-sensitive.

tablespace_name
Destination tablespace for the index.

parallel
Degree of parallelism to use when rebuilding the index.

online
`TRUE` allows DML operations affecting the index during rebuilding of the index; `FALSE` (the default) does not allow DML operations affecting the index during rebuilding of the index.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

You must have DBA privileges to call this procedure.

For an explanation of data type indexes, see Using Data Type Indexes.

For information about RDF network types and options, see RDF Networks.

Examples

The following example rebuilds the index on `xsd:string` typed literals and plain literals in the MY_TBS tablespace.

```
EXECUTE SEM_APIS.ALTER_DATATYPE_INDEX('http://www.w3.org/2001/XMLSchema#string',
command=>'REBUILD', tablespace_name=>'MY_TBS', parallel=>4);
```

15.5 SEM_APIS.ALTER_ENTAILMENT

Format

```
SEM_APIS.ALTER_ENTAILMENT(
     entailment_name IN VARCHAR2,
     command         IN VARCHAR2,
     tablespace_name IN VARCHAR2,
     parallel        IN NUMBER(38) DEFAULT NULL,
     network_owner   IN VARCHAR2 DEFAULT NULL,
     network_name    IN VARCHAR2 DEFAULT NULL);
```

> **Note:**
>
> This subprogram will be deprecated in a future release. It is recommended that you use the SEM_APIS.ALTER_INFERRED_GRAPH subprogram instead.

Description

Alters an entailment (rules index). Currently, the only action supported is to move the entailment to a specified tablespace.

Parameters

entailment_name
Name of the entailment.

command
Must be the string `MOVE`.

tablespace_name
Name of the destination tablespace.

parallel
Degree of parallelism to be associated with the operation. For more information about parallel execution, see *Oracle Database VLDB and Partitioning Guide*.

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

For an explanation of entailments, see Entailments (Rules Indexes).

For information about semantic network types and options, see RDF Networks.

Examples

The following example moves the entailment named `rdfs_rix_family` to the tablespace named `my_tbs`.

```
EEXECUTE SEM_APIS.ALTER_ENTAILMENT('rdfs_rix_family', 'MOVE',  'my_tbs');
```

15.6 SEM_APIS.ALTER_INDEX_ON_INFERRED_GRAPH

Format

```
SEM_APIS.ALTER_INDEX_ON_INFERRED_GRAPH(
     inferred_graph_name  IN VARCHAR2,
     index_code           IN VARCHAR2,
     command              IN VARCHAR2,
     tablespace_name      IN VARCHAR2 DEFAULT NULL,
     use_compression      IN BOOLEAN DEFAULT NULL,
     parallel             IN NUMBER(38) DEFAULT NULL,
     online               IN BOOLEAN DEFAULT FALSE),
     options              IN VARCHAR2 DEFAULT NULL,
     network_owner        IN VARCHAR2 DEFAULT NULL,
     network_name         IN VARCHAR2 DEFAULT NULL);
```

Description

Alters an RDF network index on an inferred graph.

Parameters

inferred_graph_name
Name of the inferred graph.

index_code
Index code string.

command
String value containing one of the following commands: `REBUILD` rebuilds the RDF network index on the inferred graph, or `UNUSABLE` marks as unusable the RDF network index on the inferred graph. The value for this parameter is not case-sensitive.

tablespace_name
Name of the destination tablespace for the rebuild operation.

use_compression
Specifies whether compression should be used when rebuilding the index.

parallel
Degree of parallelism to be associated with the operation. For more information about parallel execution, see *Oracle Database VLDB and Partitioning Guide*.

online
`TRUE` allows DML operations affecting the index during the rebuilding of the index; `FALSE` (the default) does not allow DML operations affecting the index during the rebuilding of the index.

options
(Not currently used.)

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

For an explanation of RDF network indexes, see Using Semantic Network Indexes.

For information about RDF network types and options, see RDF Networks.

Examples

The following example rebuilds (and makes usable if it is unusable) the RDF network index on the inferred graph named `rdfs_rix_family`.

```
EXECUTE SEM_APIS.ALTER_INDEX_ON_INFERRED_GRAPH('rdfs_rix_family', 'pscm', 'rebuild');
```

15.7 SEM_APIS.ALTER_INDEX_ON_RDF_GRAPH

Format

```
SEM_APIS.ALTER_INDEX_ON_RDF_GRAPH(
     rdf_graph_name    IN VARCHAR2,
     index_code        IN VARCHAR2,
     command           IN VARCHAR2,
     tablespace_name   IN VARCHAR2 DEFAULT NULL,
     use_compression   IN BOOLEAN DEFAULT NULL,
     parallel          IN NUMBER(38) DEFAULT NULL,
     online            IN BOOLEAN DEFAULT FALSE),
     options           IN VARCHAR2 DEFAULT NULL),
     network_owner     IN VARCHAR2 DEFAULT NULL,
     network_name      IN VARCHAR2 DEFAULT NULL);
```

Description

Alters an RDF network index on an RDF graph.

Parameters

rdf_graph_name
Name of the RDF graph.

index_code
Index code string.

command
String value containing one of the following commands: `REBUILD` rebuilds the RDF network index on the RDF graph, or `UNUSABLE` marks as unusable the RDF network index on the RDF graph. The value for this parameter is not case-sensitive.

tablespace_name
Name of the destination tablespace for the rebuild operation.

use_compression
Specifies whether compression should be used when rebuilding the index.

parallel
Degree of parallelism to be associated with the operation. For more information about parallel execution, see *Oracle Database VLDB and Partitioning Guide*.

online
`TRUE` allows DML operations affecting the index during the rebuilding of the index; `FALSE` (the default) does not allow DML operations affecting the index during the rebuilding of the index.

options
(Not currently used.)

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

For an explanation of RDF network indexes, see Using Semantic Network Indexes.

For information about RDF network types and options, see RDF Networks.

Examples

The following example rebuilds (and makes usable if it is unusable) the RDF network index on the RDF graph named `family`.

```
EXECUTE SEM_APIS.ALTER_INDEX_ON_RDF_GRAPH('family', 'pscm', 'rebuild');
```

15.8 SEM_APIS.ALTER_INFERRED_GRAPH

Format

```
SEM_APIS.ALTER_INFERRED_GRAPH(
     inferred_graph_name IN VARCHAR2,
     command             IN VARCHAR2,
     tablespace_name     IN VARCHAR2,
     parallel            IN NUMBER(38) DEFAULT NULL,
     network_owner       IN VARCHAR2 DEFAULT NULL,
     network_name        IN VARCHAR2 DEFAULT NULL);
```

Description

Alters an inferred graph (rules index). Currently, the only action supported is to move the inferred graph to a specified tablespace.

Parameters

inferred_graph_name
Name of the inferred graph.

command
Must be the string `MOVE`.

tablespace_name
Name of the destination tablespace.

parallel
Degree of parallelism to be associated with the operation. For more information about parallel execution, see *Oracle Database VLDB and Partitioning Guide*.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

For an explanation of inferred graphs, see Inferred Graphs.

For information about RDF network types and options, see RDF Networks.

Examples

The following example moves the inferred graph named `rdfs_rix_family` to the tablespace named `my_tbs`.

```
EEXECUTE SEM_APIS.ALTER_INFERRED_GRAPH('rdfs_rix_family', 'MOVE',  'my_tbs');
```

15.9 SEM_APIS.ALTER_MODEL

Format

```
SEM_APIS.ALTER_MODEL(
     model_name      IN VARCHAR2,
     command         IN VARCHAR2,
     tablespace_name IN VARCHAR2,
     parallel        IN NUMBER(38) DEFAULT NULL,
     network_owner   IN VARCHAR2 DEFAULT NULL,
     network_name    IN VARCHAR2 DEFAULT NULL);
```

> **Note:**
>
> This subprogram will be deprecated in a future release. It is recommended that you use the SEM_APIS.ALTER_RDF_GRAPH subprogram instead.

Description

Alters a model. Currently, the only action supported is to move the model to a specified tablespace.

Parameters

model_name
Name of the model.

command
Must be the string `MOVE`.

tablespace_name
Name of the destination tablespace.

parallel
Degree of parallelism to be associated with the operation. For more information about parallel execution, see *Oracle Database VLDB and Partitioning Guide*.

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

For an explanation of models, see Semantic Data Modeling and Semantic Data in the Database.

For information about semantic network types and options, see RDF Networks.

Examples

The following example moves the model named `family` to the tablespace named `my_tbs`.

```
EEXECUTE SEM_APIS.ALTER_MODEL('family', 'MOVE',  'my_tbs');
```

15.10 SEM_APIS.ALTER_RDF_GRAPH

Format

```
SEM_APIS.ALTER_RDF_GRAPH(
     rdf_graph_name  IN VARCHAR2,
     command         IN VARCHAR2,
     tablespace_name IN VARCHAR2,
     parallel        IN NUMBER(38) DEFAULT NULL,
     network_owner   IN VARCHAR2 DEFAULT NULL,
     network_name    IN VARCHAR2 DEFAULT NULL);
```

Description

Alters an RDF graph. Currently, the only supported action is to move the graph to a specified tablespace.

Parameters

rdf_graph_name
Name of the RDF graph.

command
Must be the string `MOVE`.

tablespace_name
Name of the destination tablespace.

parallel
Degree of parallelism to be associated with the operation. For more information about parallel execution, see *Oracle Database VLDB and Partitioning Guide*.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

- See RDF Data in the Database for more information on sematic data.
- See RDF Networks for more information about RDF network types and options.

Examples

The following example moves the RDF graph named `family` to the tablespace named `my_tbs`.

```
EEXECUTE SEM_APIS.ALTER_RDF_GRAPH('family', 'MOVE',  'my_tbs');
```

15.11 SEM_APIS.ALTER_RDF_INDEXES

Format

```
SEM_APIS.ALTER_RDF_INDEXES(
     attr_name       IN VARCHAR2,
     new_val         IN VARCHAR2,
     options         IN VARCHAR2 DEFAULT NULL,
     network_owner   IN VARCHAR2 DEFAULT NULL,
     network_name    IN VARCHAR2 DEFAULT NULL);
```

Description

Alters an attribute of all indexes on RDF_VALUE$ and RDF_LINK$ tables.

Parameters

attr_name
Attribute to be altered..

new_val
New value for the attribute.

options
(Not currently used.)

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

You must have DBA privileges to call this procedure.

Currently, the only `attr_name` value supported is `VISIBILITY`, and the only `new_val` values supported are `Y` (visible indexes) and `N` (invisible indexes).

For an explanation of RDF network indexes, see Using Semantic Network Indexes, including the subtopic about using invisible indexes.

For information about RDF network types and options, see RDF Networks.

Examples

The following example makes all RDF network indexes invisible.

```
EXECUTE SEM_APIS.ALTER_RDF_INDEXES('VISIBILITY', 'N');
```

15.12 SEM_APIS.ALTER_RESULT_TAB

Format

```
SEM_APIS.ALTER_RESULT_TAB (
     query_pattern_type IN NUMBER,
     result_tab_name    IN VARCHAR2,
     rdf_graph_name     IN VARCHAR2,
     command            IN VARCHAR2,
     pred_name          IN VARCHAR2,
     occurrence         IN NUMBER DEFAULT NULL,
     reversed           IN BOOLEAN DEFAULT FALSE,
     degree             IN NUMBER DEFAULT NULL,
     options            IN VARCHAR2 DEFAULT NULL,
     network_owner      IN DBMS_ID DEFAULT NULL,
     network_name       IN VARCHAR2 DEFAULT NULL);
```

Description

Alters the presence or extent of presence of the columns corresponding to a predicate in a given result table. Also, allows controlling the visibility of a result table to either allow or prevent its use in processing SPARQL queries against the RDF graph.

Parameters

query_pattern_type
Type of the SPM table.
The value can be one of the following:

- `SEM_APIS.SPM_TYPE_SVP`
- `SEM_APIS.SPM_TYPE_MVP`
- `SEM_APIS.SPM_TYPE_PCN`

result_tab_name
String for use as part of the name of the result table. If the target is an MVP table, then specify the name of the property.

rdf_graph_name
Name of the RDF graph.

command
Determines the type of alteration.
The supported commands are:

- **`ADD_PREDICATE`:** Adds columns for the target predicate to a result table, where the target predicate is found. Applies to SVP tables only and succeeds only if the target predicate is single-valued in the given RDF graph.
- **`DROP_PREDICATE`:** Drops columns for the target predicate from the result table, where the target predicate is found. Note that this applies to SVP tables only.
- **`ADD_VALUE`:** Adds value columns for the target predicate to a result table, where the target predicate is found.

- **DROP_VALUE**: Drops value columns for the target predicate from a result table, where the target predicate is found.
- **ADD_S_VALUE**: Includes lexical values for the subject.
- **DROP_S_VALUE**: Drops lexical values for the subject.
- **VISIBLE**: Makes the specified result table visible for possible use in processing SPARQL queries against the RDF graph. Note that this is the default setting.
- **INVISIBLE**: Makes the specified result table invisible to prevent its use in processing SPARQL queries against the RDF graph.

pred_name
Name of the target predicate if the result table is of type SVP or PCN. Must be `NULL` for an MVP type table.

occurrence
Applies only to a result table of type PCN and when the command is `ADD_VALUE` or `DROP_VALUE`.

reversed
Applies only to a result table of type SVP and when the command is `ADD_PREDICATE`.

degree
Degree of parallelism to use.

options
Reserved for future use.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

Examples

The following example adds in-line lexical value for the `<http://www.example.com#lname>` property:

```
BEGIN
  SEM_APIS.ALTER_RESULT_TAB(
     query_pattern_type => SEM_APIS.SPM_TYPE_SVP
   , result_tab_name    => 'FLHF'
   , rdf_graph_name     => 'M1'
   , command            => 'ADD_VALUE'
   , pred_name          => '<http://www.example.com#lname>'
   , network_owner      => 'RDFUSER'
   , network_name       => 'NET1'
  );
END;
/
```

15.13 SEM_APIS.ALTER_SEM_INDEX_ON_ENTAILMENT

Format

```
SEM_APIS.ALTER_SEM_INDEX_ON_ENTAILMENT(
     entailment_name  IN VARCHAR2,
     index_code       IN VARCHAR2,
     command          IN VARCHAR2,
     tablespace_name  IN VARCHAR2 DEFAULT NULL,
     use_compression  IN BOOLEAN DEFAULT NULL,
     parallel         IN NUMBER(38) DEFAULT NULL,
     online           IN BOOLEAN DEFAULT FALSE),
     options          IN VARCHAR2 DEFAULT NULL,
     network_owner    IN VARCHAR2 DEFAULT NULL,
     network_name     IN VARCHAR2 DEFAULT NULL);
```

> **Note:**
>
> This subprogram will be deprecated in a future release. It is recommended that you use the SEM_APIS.ALTER_INDEX_ON_INFERRED_GRAPH subprogram instead.

Description

Alters a semantic network index on an entailment.

Parameters

entailment_name
Name of the entailment.

index_code
Index code string.

command
String value containing one of the following commands: `REBUILD` rebuilds the semantic network index on the entailment, or `UNUSABLE` marks as unusable the semantic network index on the entailment. The value for this parameter is not case-sensitive.

tablespace_name
Name of the destination tablespace for the rebuild operation.

use_compression
Specifies whether compression should be used when rebuilding the index.

parallel
Degree of parallelism to be associated with the operation. For more information about parallel execution, see *Oracle Database VLDB and Partitioning Guide*.

online
`TRUE` allows DML operations affecting the index during the rebuilding of the index; `FALSE` (the default) does not allow DML operations affecting the index during the rebuilding of the index.

options
(Not currently used.)

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

For an explanation of semantic network indexes, see Using Semantic Network Indexes.

For information about semantic network types and options, see RDF Networks.

Examples

The following example rebuilds (and makes usable if it is unusable) the semantic network index on the entailment named `rdfs_rix_family`.

```
EXECUTE SEM_APIS.ALTER_SEM_INDEX_ON_ENTAILMENT('rdfs_rix_family', 'pscm', 'rebuild');
```

15.14 SEM_APIS.ALTER_SEM_INDEX_ON_MODEL

Format

```
SEM_APIS.ALTER_SEM_INDEX_ON_MODEL(
     model_name        IN VARCHAR2,
     index_code        IN VARCHAR2,
     command           IN VARCHAR2,
     tablespace_name   IN VARCHAR2 DEFAULT NULL,
     use_compression   IN BOOLEAN DEFAULT NULL,
     parallel          IN NUMBER(38) DEFAULT NULL,
     online            IN BOOLEAN DEFAULT FALSE),
     options           IN VARCHAR2 DEFAULT NULL),
     network_owner     IN VARCHAR2 DEFAULT NULL,
     network_name      IN VARCHAR2 DEFAULT NULL);
```

> **Note:**
>
> This subprogram will be deprecated in a future release. It is recommended that you use the SEM_APIS.ALTER_INDEX_ON_RDF_GRAPH subprogram instead.

Description

Alters a semantic network index on a model.

Parameters

model_name
Name of the model.

index_code
Index code string.

command
String value containing one of the following commands: `REBUILD` rebuilds the semantic network index on the model, or `UNUSABLE` marks as unusable the semantic network index on the model. The value for this parameter is not case-sensitive.

tablespace_name
Name of the destination tablespace for the rebuild operation.

use_compression
Specifies whether compression should be used when rebuilding the index.

parallel
Degree of parallelism to be associated with the operation. For more information about parallel execution, see *Oracle Database VLDB and Partitioning Guide*.

online
`TRUE` allows DML operations affecting the index during the rebuilding of the index; `FALSE` (the default) does not allow DML operations affecting the index during the rebuilding of the index.

options
(Not currently used.)

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

For an explanation of semantic network indexes, see Using Semantic Network Indexes.

For information about semantic network types and options, see RDF Networks.

Examples

The following example rebuilds (and makes usable if it is unusable) the semantic network index on the model named `family`.

```
EXECUTE SEM_APIS.ALTER_SEM_INDEX_ON_MODEL('family', 'pscm', 'rebuild');
```

15.15 SEM_APIS.ALTER_SEM_INDEXES

Format

```
SEM_APIS.ALTER_SEM_INDEXES(
     attr_name  IN VARCHAR2,
     new_val    IN VARCHAR2,
     options    IN VARCHAR2 DEFAULT NULL,
     network_owner   IN VARCHAR2 DEFAULT NULL,
     network_name    IN VARCHAR2 DEFAULT NULL);
```

> **Note:**
>
> This subprogram will be deprecated in a future release. It is recommended that you use the SEM_APIS.ALTER_RDF_INDEXES subprogram instead.

Description

Alters an attribute of all indexes on RDF_VALUE$ and RDF_LINK$ tables.

Parameters

attr_name
Attribute to be altered..

new_val
New value for the attribute.

options
(Not currently used.)

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

You must have DBA privileges to call this procedure.

Currently, the only `attr_name` value supported is `VISIBILITY`, and the only `new_val` values supported are `Y` (visible indexes) and `N` (invisible indexes).

For an explanation of semantic network indexes, see Using Semantic Network Indexes, including the subtopic about using invisible indexes.

For information about semantic network types and options, see RDF Networks.

Examples

The following example makes all semantic network indexes invisible.

```
EXECUTE SEM_APIS.ALTER_SEM_INDEXES('VISIBILITY', 'N');
```

15.16 SEM_APIS.ALTER_SPM_TAB

Format

```
SEM_APIS.ALTER_SPM_TAB (
     spm_type            IN NUMBER,
     spm_name            IN DBMS_ID,
     model_name          IN VARCHAR2,
     command             IN VARCHAR2,
     pred_name           IN VARCHAR2,
     occurrence          IN NUMBER DEFAULT NULL,
     reversed            IN BOOLEAN DEFAULT FALSE,
     degree              IN NUMBER DEFAULT NULL,
     options             IN VARCHAR2 DEFAULT NULL,
     network_owner       IN DBMS_ID DEFAULT NULL,
     network_name        IN VARCHAR2 DEFAULT NULL);
```

> **Note:**
>
> This subprogram will be deprecated in a future release. It is recommended that you use the SEM_APIS.ALTER_RESULT_TAB subprogram instead.

Description

Alters the presence or extent of presence of the columns corresponding to a predicate in a given SPM table.

Parameters

spm_type
Type of the SPM table.
The value can be one of the following:

- `SEM_APIS.SPM_TYPE_SVP`
- `SEM_APIS.SPM_TYPE_MVP`
- `SEM_APIS.SPM_TYPE_PCN`

spm_name
String for use as part of the name of the SPM table. If the target is an MVP table, then specify the name of the property.

model_name
Name of the RDF model.

command
Determines the type of alteration.
The supported commands are:

- **`ADD_PREDICATE:`** Adds columns for the target predicate to an SPM table, where the target predicate is found. Applies to SVP tables only and succeeds only if the target predicate is single-valued in the given RDF model.
- **`DROP_PREDICATE:`** Drops columns for the target predicate from the SPM table, where the target predicate is found. Note that this applies to SVP tables only.
- **`ADD_VALUE:`** Adds value columns for the target predicate to an SPM table, where the target predicate is found.
- **`DROP_VALUE:`** Drops value columns for the target predicate from an SPM table, where the target predicate is found.
- **`ADD_S_VALUE:`** Includes lexical values for the subject.
- **`DROP_S_VALUE:`** Drops lexical values for the subject.

pred_name
Name of the target predicate if the SPM table is of type SVP or PCN. Must be `NULL` for an MVP type table.

occurrence
Applies only to an SPM table of type PCN and when the command is `ADD_VALUE` or `DROP_VALUE`.

reversed
Applies only to an SPM table of type SVP and when the command is `ADD_PREDICATE`.

degree
Degree of parallelism to use.

options
Reserved for future use.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

Examples

The following example adds in-line lexical value for the `<http://www.example.com#lname>` property:

```
BEGIN
  SEM_APIS.ALTER_SPM_TAB(
     spm_type      => SEM_APIS.SPM_TYPE_SVP
   , spm_name      => 'FLHF'
   , model_name    => 'M1'
   , command       => 'ADD_VALUE'
   , pred_name     => '<http://www.example.com#lname>'
   , network_owner => 'RDFUSER'
   , network_name  => 'NET1'
  );
END;
/
```

15.17 SEM_APIS.ANALYZE_ENTAILMENT

Format

```
SEM_APIS.ANALYZE_ENTAILMENT(
     entailment_name  IN VARCHAR2,
     estimate_percent IN NUMBER DEFAULT to_estimate_percent_type
(get_param('ESTIMATE_PERCENT')),
     method_opt       IN VARCHAR2 DEFAULT get_param('METHOD_OPT'),
     degree           IN NUMBER DEFAULT to_degree_type(get_param('DEGREE')),
     cascade          IN BOOLEAN DEFAULT to_cascade_type(get_param('CASCADE')),
     no_invalidate    IN BOOLEAN DEFAULT to_no_invalidate_type
(get_param('NO_INVALIDATE')),
     force            IN BOOLEAN DEFAULT FALSE),
     network_owner    IN VARCHAR2 DEFAULT NULL,
     network_name     IN VARCHAR2 DEFAULT NULL);
```

> **Note:**
>
> This subprogram will be deprecated in a future release. It is recommended that you use the SEM_APIS.ANALYZE_INFERRED_GRAPH subprogram instead.

Description

Collects statistics for a specified entailment (rules index).

Parameters

entailment_name
Name of the entailment.

estimate_percent
Percentage of rows to estimate in the internal table partition containing information about the entailment (`NULL` means compute). The valid range is [0.000001,100]. Use the constant `DBMS_STATS.AUTO_SAMPLE_SIZE` to have Oracle determine the appropriate sample size for good statistics. This is the usual default.

method_opt
Accepts either of the following options, or both in combination, for the internal table partition containing information about the entailment:

- `FOR ALL [INDEXED | HIDDEN] COLUMNS [size_clause]`
- `FOR COLUMNS` [size clause] column|attribute [size_clause] [,column|attribute [size_clause]...]

`size_clause` is defined as `size_clause := SIZE {integer | REPEAT | AUTO | SKEWONLY}`
`column` is defined as `column := column_name | (extension)`

- `integer` : Number of histogram buckets. Must be in the range [1,254].
- `REPEAT` : Collects histograms only on the columns that already have histograms.
- `AUTO` : Oracle determines the columns to collect histograms based on data distribution and the workload of the columns.
- `SKEWONLY` : Oracle determines the columns to collect histograms based on the data distribution of the columns.
- `column_name` : name of a column
- `extension:` Can be either a column group in the format of (`column_name`, `column_name` [, ...]) or an expression.

The usual default is `FOR ALL COLUMNS SIZE AUTO`.

degree
Degree of parallelism for the internal table partition containing information about the entailment. The usual default for `degree` is `NULL`, which means use the table default value specified by the `DEGREE` clause in the `CREATE TABLE` or `ALTER TABLE` statement. Use the constant `DBMS_STATS.DEFAULT_DEGREE` to specify the default value based on the initialization parameters. The `AUTO_DEGREE` value determines the degree of parallelism automatically. This is either 1 (serial execution) or `DEFAULT_DEGREE` (the system default value based on number of CPUs and initialization parameters) according to size of the object.

cascade
Gathers statistics on the indexes for the internal table partition containing information about the entailment. Use the constant `DBMS_STATS.AUTO_CASCADE` to have Oracle determine whether index statistics are to be collected or not. This is the usual default.

no_invalidate
Does not invalidate the dependent cursors if set to `TRUE`. The procedure invalidates the dependent cursors immediately if set to `FALSE`. Use `DBMS_STATS.AUTO_INVALIDATE`. to have Oracle decide when to invalidate dependent cursors. This is the usual default.

force
`TRUE` gathers statistics even if the entailment is locked; `FALSE` (the default) does not gather statistics if the entailment is locked.

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

Index statistics collection can be parellelized except for cluster, domain, and join indexes.

This procedure internally calls the DBMS_STATS.GATHER_TABLE_STATS procedure, which collects statistics for the internal table partition that contains information about the entailment. The DBMS_STATS.GATHER_TABLE_STATS procedure is documented in *Oracle Database PL/SQL Packages and Types Reference*.

See also Managing Statistics for Semantic Models and the Semantic Network.

For information about entailments, see Entailments (Rules Indexes).

For information about semantic network types and options, see RDF Networks.

Examples

The following example collects statistics for the entailment named `rdfs_rix_family`.

```
EXECUTE SEM_APIS.ANALYZE_ENTAILMENT('rdfs_rix_family');
```

15.18 SEM_APIS.ANALYZE_INFERRED_GRAPH

Format

```
SEM_APIS.ANALYZE_INFERRED_GRAPH(
     inferred_graph_name   IN VARCHAR2,
     estimate_percent      IN NUMBER DEFAULT to_estimate_percent_type
(get_param('ESTIMATE_PERCENT')),
     method_opt            IN VARCHAR2 DEFAULT get_param('METHOD_OPT'),
     degree                IN NUMBER DEFAULT to_degree_type(get_param('DEGREE')),
     cascade               IN BOOLEAN DEFAULT to_cascade_type(get_param('CASCADE')),
     no_invalidate         IN BOOLEAN DEFAULT to_no_invalidate_type
(get_param('NO_INVALIDATE')),
     force                 IN BOOLEAN DEFAULT FALSE),
     network_owner         IN VARCHAR2 DEFAULT NULL,
     network_name          IN VARCHAR2 DEFAULT NULL);
```

Description

Collects statistics for a specified inferred graph (rules index).

Parameters

inferred_graph_name
Name of the inferred graph.

estimate_percent
Percentage of rows to estimate in the internal table partition containing information about the inferred graph (`NULL` means compute). The valid range is [0.000001,100]. Use the constant `DBMS_STATS.AUTO_SAMPLE_SIZE` to have Oracle determine the appropriate sample size for good statistics. This is the usual default.

method_opt
Accepts either of the following options, or both in combination, for the internal table partition containing information about the inferred graph:

- `FOR ALL [INDEXED | HIDDEN] COLUMNS [size_clause]`
- `FOR COLUMNS` [size clause] column|attribute [size_clause] [,column|attribute [size_clause]...]

`size_clause` is defined as `size_clause := SIZE {integer | REPEAT | AUTO | SKEWONLY}`
`column` is defined as `column := column_name | (extension)`

- `integer` : Number of histogram buckets. Must be in the range [1,254].
- `REPEAT` : Collects histograms only on the columns that already have histograms.
- `AUTO` : Oracle determines the columns to collect histograms based on data distribution and the workload of the columns.
- `SKEWONLY` : Oracle determines the columns to collect histograms based on the data distribution of the columns.
- `column_name` : name of a column
- `extension:` Can be either a column group in the format of (`column_name`, `column_name` [, ...]) or an expression.

The usual default is `FOR ALL COLUMNS SIZE AUTO`.

degree
Degree of parallelism for the internal table partition containing information about the inferred graph. The usual default for `degree` is `NULL`, which means use the table default value specified by the `DEGREE` clause in the `CREATE TABLE` or `ALTER TABLE` statement. Use the constant `DBMS_STATS.DEFAULT_DEGREE` to specify the default value based on the initialization parameters. The `AUTO_DEGREE` value determines the degree of parallelism automatically. This is either 1 (serial execution) or `DEFAULT_DEGREE` (the system default value based on number of CPUs and initialization parameters) according to size of the object.

cascade
Gathers statistics on the indexes for the internal table partition containing information about the inferred graph. Use the constant `DBMS_STATS.AUTO_CASCADE` to have Oracle determine whether index statistics are to be collected or not. This is the usual default.

no_invalidate
Does not invalidate the dependent cursors if set to `TRUE`. The procedure invalidates the dependent cursors immediately if set to `FALSE`. Use `DBMS_STATS.AUTO_INVALIDATE`. to have Oracle decide when to invalidate dependent cursors. This is the usual default.

force
`TRUE` gathers statistics even if the inferred graph is locked; `FALSE` (the default) does not gather statistics if the inferred graph is locked.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

Index statistics collection can be parellelized except for cluster, domain, and join indexes.

This procedure internally calls the DBMS_STATS.GATHER_TABLE_STATS procedure, which collects statistics for the internal table partition that contains information about the inferred graph. The DBMS_STATS.GATHER_TABLE_STATS procedure is documented in *Oracle Database PL/SQL Packages and Types Reference*.

See also Managing Statistics for the RDF Graphs and RDF Network.

For information about inferred graphs, see Inferred Graphs.

For information about RDF network types and options, see RDF Networks.

Examples

The following example collects statistics for the inferred graph named `rdfs_rix_family`.

```
EXECUTE SEM_APIS.ANALYZE_INFERRED_GRAPH('rdfs_rix_family');
```

15.19 SEM_APIS.ANALYZE_MODEL

Format

```
SEM_APIS.ANALYZE_MODEL(
     model_name       IN VARCHAR2,
     estimate_percent IN NUMBER DEFAULT to_estimate_percent_type
(get_param('ESTIMATE_PERCENT')),
     method_opt       IN VARCHAR2 DEFAULT get_param('METHOD_OPT'),
     degree           IN NUMBER DEFAULT to_degree_type(get_param('DEGREE')),
     cascade          IN BOOLEAN DEFAULT to_cascade_type(get_param('CASCADE')),
     no_invalidate    IN BOOLEAN DEFAULT to_no_invalidate_type
(get_param('NO_INVALIDATE')),
     force            IN BOOLEAN DEFAULT FALSE),
     network_owner    IN VARCHAR2 DEFAULT NULL,
     network_name     IN VARCHAR2 DEFAULT NULL);
```

> **Note:**
>
> This subprogram will be deprecated in a future release. It is recommended that you use the SEM_APIS.ANALYZE_RDF_GRAPH subprogram instead.

Description

Collects optimizer statistics for a specified model.

Parameters

model_name
Name of the model.

estimate_percent
Percentage of rows to estimate in the internal table partition containing information about the model (`NULL` means compute). The valid range is [0.000001,100]. Use the constant `DBMS_STATS.AUTO_SAMPLE_SIZE` to have Oracle determine the appropriate sample size for good statistics. This is the usual default.

method_opt
Accepts either of the following options, or both in combination, for the internal table partition containing information about the model:

- `FOR ALL [INDEXED | HIDDEN] COLUMNS [size_clause]`
- `FOR COLUMNS` [size clause] column|attribute [size_clause] [,column|attribute [size_clause]...]

`size_clause` is defined as `size_clause := SIZE {integer | REPEAT | AUTO | SKEWONLY}`
`column` is defined as `column := column_name | (extension)`

- `integer` : Number of histogram buckets. Must be in the range [1,254].
- `REPEAT` : Collects histograms only on the columns that already have histograms.
- `AUTO` : Oracle determines the columns to collect histograms based on data distribution and the workload of the columns.
- `SKEWONLY` : Oracle determines the columns to collect histograms based on the data distribution of the columns.
- `column_name` : name of a column
- `extension:` Can be either a column group in the format of (`column_name`, `column_name` [, ...]) or an expression.

The usual default is `FOR ALL COLUMNS SIZE AUTO`.

degree
Degree of parallelism for the internal table partition containing information about the model. The usual default for `degree` is `NULL`, which means use the table default value specified by the `DEGREE` clause in the `CREATE TABLE` or `ALTER TABLE` statement. Use the constant `DBMS_STATS.DEFAULT_DEGREE` to specify the default value based on the initialization parameters. The `AUTO_DEGREE` value determines the degree of parallelism automatically. This is either 1 (serial execution) or `DEFAULT_DEGREE` (the system default value based on number of CPUs and initialization parameters) according to size of the object.

cascade
Gathers statistics on the indexes for the internal table partition containing information about the model. Use the constant `DBMS_STATS.AUTO_CASCADE` to have Oracle determine whether index statistics are to be collected or not. This is the usual default.

no_invalidate
Does not invalidate the dependent cursors if set to `TRUE`. The procedure invalidates the dependent cursors immediately if set to `FALSE`. Use `DBMS_STATS.AUTO_INVALIDATE`. to have Oracle decide when to invalidate dependent cursors. This is the usual default.

force
`TRUE` gathers statistics even if the model is locked; `FALSE` (the default) does not gather statistics if the model is locked.

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

Index statistics collection can be parellelized except for cluster, domain, and join indexes.

This procedure internally calls the DBMS_STATS.GATHER_TABLE_STATS procedure, which collects optimizer statistics for the internal table partition that contains information about the model. The DBMS_STATS.GATHER_TABLE_STATS procedure is documented in *Oracle Database PL/SQL Packages and Types Reference*.

See also Managing Statistics for Semantic Models and the Semantic Network.

For information about semantic network types and options, see RDF Networks.

Examples

The following example collects statistics for the semantic model named `family`.

```
EXECUTE SEM_APIS.ANALYZE_MODEL('family');
```

15.20 SEM_APIS.ANALYZE_RDF_GRAPH

Format

```
SEM_APIS.ANALYZE_RDF_GRAPH(
     rdf_graph_name    IN VARCHAR2,
     estimate_percent IN NUMBER DEFAULT to_estimate_percent_type
(get_param('ESTIMATE_PERCENT')),
     method_opt        IN VARCHAR2 DEFAULT get_param('METHOD_OPT'),
     degree            IN NUMBER DEFAULT to_degree_type(get_param('DEGREE')),
     cascade           IN BOOLEAN DEFAULT to_cascade_type(get_param('CASCADE')),
     no_invalidate     IN BOOLEAN DEFAULT to_no_invalidate_type
(get_param('NO_INVALIDATE')),
     force             IN BOOLEAN DEFAULT FALSE),
     network_owner     IN VARCHAR2 DEFAULT NULL,
     network_name      IN VARCHAR2 DEFAULT NULL);
```

Description

Collects optimizer statistics for a specified RDF graph.

Parameters

rdf_graph_name
Name of the RDF graph.

estimate_percent
Percentage of rows to estimate in the internal table partition containing information about the RDF graph (`NULL` means compute). The valid range is [0.000001,100]. Use the constant `DBMS_STATS.AUTO_SAMPLE_SIZE` to have Oracle determine the appropriate sample size for good statistics. This is the usual default.

method_opt
Accepts either of the following options, or both in combination, for the internal table partition containing information about the RDF graph:

- `FOR ALL [INDEXED | HIDDEN] COLUMNS [size_clause]`
- `FOR COLUMNS` [size clause] column|attribute [size_clause] [,column|attribute [size_clause]...]

`size_clause` is defined as `size_clause := SIZE {integer | REPEAT | AUTO | SKEWONLY}`
`column` is defined as `column := column_name | (extension)`

- `integer` : Number of histogram buckets. Must be in the range [1,254].
- `REPEAT` : Collects histograms only on the columns that already have histograms.
- `AUTO` : Oracle determines the columns to collect histograms based on data distribution and the workload of the columns.
- `SKEWONLY` : Oracle determines the columns to collect histograms based on the data distribution of the columns.
- `column_name` : name of a column
- `extension:` Can be either a column group in the format of (`column_name`, `column_name` [, ...]) or an expression.

The usual default is `FOR ALL COLUMNS SIZE AUTO`.

degree
Degree of parallelism for the internal table partition containing information about the RDF graph. The usual default for `degree` is `NULL`, which means use the table default value specified by the `DEGREE` clause in the `CREATE TABLE` or `ALTER TABLE` statement. Use the constant `DBMS_STATS.DEFAULT_DEGREE` to specify the default value based on the initialization parameters. The `AUTO_DEGREE` value determines the degree of parallelism automatically. This is either 1 (serial execution) or `DEFAULT_DEGREE` (the system default value based on number of CPUs and initialization parameters) according to size of the object.

cascade
Gathers statistics on the indexes for the internal table partition containing information about the RDF graph. Use the constant `DBMS_STATS.AUTO_CASCADE` to have Oracle determine whether index statistics are to be collected or not. This is the usual default.

no_invalidate
Does not invalidate the dependent cursors if set to `TRUE`. The procedure invalidates the dependent cursors immediately if set to `FALSE`. Use `DBMS_STATS.AUTO_INVALIDATE`. to have Oracle decide when to invalidate dependent cursors. This is the usual default.

force
`TRUE` gathers statistics even if the RDF graph is locked; `FALSE` (the default) does not gather statistics if the graph is locked.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

Index statistics collection can be parellelized except for cluster, domain, and join indexes.

This procedure internally calls the DBMS_STATS.GATHER_TABLE_STATS procedure, which collects optimizer statistics for the internal table partition that contains information about the RDF graph. The DBMS_STATS.GATHER_TABLE_STATS procedure is documented in *Oracle Database PL/SQL Packages and Types Reference*.

See also Managing Statistics for Semantic Models and the Semantic Network.

For information about RDF network types and options, see RDF Networks.

Examples

The following example collects statistics for the RDF graph named `family`.

```
EXECUTE SEM_APIS.ANALYZE_RDF_GRAPH('family');
```

15.21 SEM_APIS.APPEND_RDF_NETWORK_DATA

Format

```
SEM_APIS.APPEND_RDF_NETWORK_DATA(
     from_schema    IN DBMS_ID,
     degree         IN INTEGER DEFAULT NULL,
     options        IN VARCHAR2 DEFAULT NULL,
     network_owner  IN VARCHAR2 DEFAULT NULL,
     network_name   IN VARCHAR2 DEFAULT NULL);
```

Description

Appends moved RDF network data from a staging schema into an RDF network.

Parameters

from_schema
The staging schema that contains moved RDF network data to be appended.

degree
Degree of parallelism to use for any SQL insert or index building operations. The default is no parallel execution.

options
String specifying any options to use during the append operation. Supported options are:

- PURGE=T – drop all remaining RDF network data in the staging schema after the append operation completes.

network_owner
Owner of the destination RDF network for the append operation. (See Table 1-2.)

network_name
Name of the destination RDF network for the append operation. (See Table 1-2.)

Usage Notes

Partition exchange operations rather than SQL INSERT statements are used to move most of the data during the append operation, so the staging schema will no longer contain complete RDF network data after the operation is complete.

You must have DBA privileges to call this procedure.

For more information and examples, see Moving, Restoring, and Appending an RDF Network.

For information about RDF network types and options, see RDF Networks.

Examples

The following example appends an RDF network from the RDFEXPIMPU staging schema into the MYNET RDF network owned by RDFADMIN.

```
EXECUTE
sem_apis.append_rdf_network_data(from_schema=>'RDFEXPIMPU',network_owner=>'RDFADMIN',netw
ork_name=>'MYNET');
```

15.22 SEM_APIS.APPEND_SEM_NETWORK_DATA

Format

```
SEM_APIS.APPEND_SEM_NETWORK_DATA(
     from_schema     IN DBMS_ID,
     degree          IN INTEGER DEFAULT NULL,
     options         IN VARCHAR2 DEFAULT NULL,
     network_owner   IN VARCHAR2 DEFAULT NULL,
     network_name    IN VARCHAR2 DEFAULT NULL);
```

> **Note:**
>
> This subprogram will be deprecated in a future release. It is recommended that you use the SEM_APIS.APPEND_RDF_NETWORK_DATA subprogram instead.

Description

Appends moved semantic network data from a staging schema into a semantic network.

Parameters

from_schema
The staging schema that contains moved semantic network data to be appended.

degree
Degree of parallelism to use for any SQL insert or index building operations. The default is no parallel execution.

options
String specifying any options to use during the append operation. Supported options are:

- PURGE=T – drop all remaining semantic network data in the staging schema after the append operation completes.

network_owner
Owner of the destination semantic network for the append operation. (See Table 1-2.)

network_name
Name of the destination semantic network for the append operation. (See Table 1-2.)

Usage Notes

Partition exchange operations rather than SQL INSERT statements are used to move most of the data during the append operation, so the staging schema will no longer contain complete semantic network data after the operation is complete.

You must have DBA privileges to call this procedure.

For more information and examples, see Moving, Restoring, and Appending an RDF Network.

For information about semantic network types and options, see RDF Networks.

Examples

The following example appends a semantic network from the RDFEXPIMPU staging schema into the MYNET semantic network owned by RDFADMIN.

```
EXECUTE
sem_apis.append_sem_network_data(from_schema=>'RDFEXPIMPU',network_owner=>'RDFADMIN',netw
ork_name=>'MYNET');
```

15.23 SEM_APIS.BUILD_RESULT_TAB

Format

```
SEM_APIS.BUILD_RESULT_TAB (
    query_pattern_type IN NUMBER,
    result_tab_name    IN VARCHAR2,
    rdf_graph_name     IN VARCHAR2,
    key_string         IN VARCHAR2,
    prefixes           IN VARCHAR2 DEFAULT NULL,
    tablespace_name    IN DBMS_ID DEFAULT NULL,
    degree             IN NUMBER DEFAULT NULL,
    options            IN VARCHAR2 DEFAULT NULL,
    network_owner      IN DBMS_ID DEFAULT NULL,
    network_name       IN VARCHAR2 DEFAULT NULL);
```

Description

Creates a result table of the specified type for the specified RDF graph.

More information on these parameters are described in the following Parameters section.

Parameters

query_pattern_type
Type of the result table.
The value can be one of the following:

- `SEM_APIS.SPM_TYPE_SVP`
- `SEM_APIS.SPM_TYPE_MVP`
- `SEM_APIS.SPM_TYPE_PCN`

result_tab_name
String for use as part of the name of the result table.
Must be `NULL` for an MVP table because the name is auto-generated as `Id(<property>)` for the (single) property specified in the `key_string` parameter.

rdf_graph_name
Name of the RDF graph.

key_string
Specifies the list of properties to be included. Each of the properties must be single-valued based on the data in the RDF graph. If a property is preceded by a '`+`' then the table will

include the columns for the lexical values. To include a reversed property, use a '^' before the property. Use of `+^:fatherOf`, for example, includes lexical value information for the reversed property (intuitively equivalent to a `:hasFather` property with lexical value information).

prefixes
Specifies the prefixes relevant to properties used in the `key_string` parameter. Syntax is same as `PREFIX` syntax used in SPARQL queries.

tablespace_name
Name of the target tablespace for the result table.

degree
Degree of parallelism to use during the operation.

options
String specifying any options to use during the operation.
Supported option is:

- `INMEMORY=T`: Builds the in-memory SVP table with all predicates or in-memory MVP table.
- `S_INDEX=F`: Skips creation of the nonunique index on the START_NODE_ID column of MVP and PCN tables.
- `P_INDEXES=F`: Skips creation of the nonunique indexes on the individual property columns of PCN tables.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

- This operation has a DDL semantics.
- The invoker must be the owner of the RDF graph or the RDF network or both.

Examples

The following example creates an SVP table:

```
BEGIN
  SEM_APIS.BUILD_RESULT_TAB(
     query_pattern_type => SEM_APIS.SPM_TYPE_SVP
   , result_tab_name    => 'FLHF'
   , rdf_graph_name     => 'M1'
   , key_string         => ' :fname :lname :height ^:fatherOf '
   , prefixes           => ' PREFIX : <http://www.example.com#> '
   , network_owner      => 'RDFUSER'
   , network_name       => 'NET1'
  );
END;
/
```

The following example creates a PCN table:

```
BEGIN
  SEM_APIS.BUILD_RESULT_TAB(
     result_tab_name    => 'GRANDPA'
   , query_pattern_type => SEM_APIS.SPM_TYPE_PCN
   , rdf_graph_name     => 'M1'
```

```
   , key_string         => ' S :fatherOf :fatherOf '
   , prefixes           => ' PREFIX : <http://www.example.com#> '
   , network_owner      => 'RDFUSER'
   , network_name       => 'NET1'
  );
END;
/
```

The following example creates an MVP table:

```
BEGIN
  SEM_APIS.BUILD_RESULT_TAB(
     query_pattern_type => SEM_APIS.SPM_TYPE_MVP
   , result_tab_name    => null /* must be NULL (the name is auto-generated based on
id(property) */
   , rdf_graph_name     => 'M1'
   , key_string         => ' :motherOf ' /* must have exactly one property */
   , prefixes           => ' PREFIX : <http://www.example.com#> '
   , network_owner      => 'RDFUSER'
   , network_name       => 'NET1'
  );
END;
/
```

15.24 SEM_APIS.BUILD_SPM_TAB

Format

```
SEM_APIS.BUILD_SPM_TAB (
    spm_type            IN NUMBER,
    spm_name            IN DBMS_ID,
    model_name          IN VARCHAR2,
    key_string          IN VARCHAR2,
    prefixes            IN VARCHAR2 DEFAULT NULL,
    tablespace_name     IN DBMS_ID DEFAULT NULL,
    degree              IN NUMBER DEFAULT NULL,
    options             IN VARCHAR2 DEFAULT NULL,
    network_owner       IN DBMS_ID DEFAULT NULL,
    network_name        IN VARCHAR2 DEFAULT NULL);
```

> **Note:**
>
> This subprogram will be deprecated in a future release. It is recommended that you use the SEM_APIS.BUILD_RESULT_TAB subprogram instead.

Description

Creates an SPM table of the specified type for the specified RDF model.

More information on these parameters are described in the following Parameters section.

Parameters

spm_type
Type of the SPM table.
The value can be one of the following:

- `SEM_APIS.SPM_TYPE_SVP`
- `SEM_APIS.SPM_TYPE_MVP`
- `SEM_APIS.SPM_TYPE_PCN`

spm_name
String for use as part of the name of the SPM table.
Must be `NULL` for an MVP table because the name is auto-generated as `Id(<property>)` for the (single) property specified in the `key_string` parameter.

model_name
Name of the RDF model.

key_string
Specifies the list of properties to be included. Each of the properties must be single-valued based on the data in the RDF model. If a property is preceded by a '`+`' then the table will include the columns for the lexical values. To include a reversed property, use a '`^`' before the property. Use of `+^:fatherOf`, for example, includes lexical value information for the reversed property (intuitively equivalent to a `:hasFather` property with lexical value information).

prefixes
Specifies the prefixes relevant to properties used in the `key_string` parameter. Syntax is same as `PREFIX` syntax used in SPARQL queries.

tablespace_name
Name of the target tablespace for the SPM table.

degree
Degree of parallelism to use during the operation.

options
String specifying any options to use during the operation.
Supported option is:

- `INMEMORY=T`: Builds the in-memory SVP table with all predicates or in-memory MVP table.
- `S_INDEX=F`: Skips creation of the nonunique index on the START_NODE_ID column of MVP and PCN tables.
- `P_INDEXES=F`: Skips creation of the nonunique indexes on the individual property columns of PCN tables.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

- This operation has a DDL semantics.
- The invoker must be the owner of the RDF model or the RDF network or both.

Examples

The following example creates an SVP table:

```
BEGIN
  SEM_APIS.BUILD_SPM_TAB(
```

```
      spm_type      => SEM_APIS.SPM_TYPE_SVP
    , spm_name      => 'FLHF'
    , model_name    => 'M1'
    , key_string    => ' :fname :lname :height ^:fatherOf '
    , prefixes      => ' PREFIX : <http://www.example.com#> '
    , network_owner => 'RDFUSER'
    , network_name  => 'NET1'
  );
END;
/
```

The following example creates a PCN table:

```
BEGIN
  SEM_APIS.BUILD_SPM_TAB(
      spm_name      => 'GRANDPA'
    , spm_type      => SEM_APIS.SPM_TYPE_PCN
    , model_name    => 'M1'
    , key_string    => ' S :fatherOf :fatherOf '
    , prefixes      => ' PREFIX : <http://www.example.com#> '
    , network_owner => 'RDFUSER'
    , network_name  => 'NET1'
  );
END;
/
```

The following example creates an MVP table:

```
BEGIN
  SEM_APIS.BUILD_SPM_TAB(
      spm_type      => SEM_APIS.SPM_TYPE_MVP
    , spm_name      => null /* must be NULL (the name is auto-generated based on
id(property) */
    , model_name    => 'M1'
    , key_string    => ' :motherOf ' /* must have exactly one property */
    , prefixes      => ' PREFIX : <http://www.example.com#> '
    , network_owner => 'RDFUSER'
    , network_name  => 'NET1'
  );
END;
/
```

15.25 SEM_APIS.BULK_LOAD_FROM_STAGING_TABLE

Format

```
SEM_APIS.BULK_LOAD_FROM_STAGING_TABLE(
     model_name    IN VARCHAR2,
     table_owner   IN VARCHAR2,
     table_name    IN VARCHAR2,
     flags         IN VARCHAR2 DEFAULT NULL,
     debug         IN INTEGER DEFAULT NULL,
     start_comment IN VARCHAR2 DEFAULT NULL,
     end_comment   IN VARCHAR2 DEFAULT NULL,
     network_owner IN VARCHAR2 DEFAULT NULL,
     network_name  IN VARCHAR2 DEFAULT NULL);
```

> **Note:**
>
> This subprogram will be deprecated in a future release. It is recommended that you use the SEM_APIS.BULK_LOAD_RDF_GRAPH subprogram instead.

Description

Loads semantic data from a staging table.

Parameters

model_name
Name of the model.

table_owner
Name of the schema that owns the staging table that holds semantic data to be loaded.

table_name
Name of the staging table that holds semantic data to be loaded.

flags
An optional quoted string with one or more of the following keyword specifications:

- `COMPRESS=CSCQH` uses COLUMN STORE COMPRESS FOR QUERY HIGH on the RDF_LINK$ partition for the model.
- `COMPRESS=CSCQL` uses COLUMN STORE COMPRESS FOR QUERY LOW on the RDF_LINK$ partition for the model.
- COMPRESS=RSCA uses ROW STORE COMPRESS ADVANCED on the RDF_LINK$ partition for the model.
- COMPRESS=RSCAB uses ROW STORE COMPRESS BASIC on the RDF_LINK$ partition for the model.
- `DEL_BATCH_DUPS=USE_INSERT` allows the use of an insertion-based strategy for duplicate elimination that may lead to faster processing if the input data contains many duplicates.
- `MBV_METHOD=SHADOW` allows the use of a different value loading strategy that may lead to faster processing for large loads.
- `PARALLEL_CREATE_INDEX` allows internal indexes to be created in parallel, which may improve the performance of the bulk load processing.
- `PARALLEL=<integer>` allows much of the processing used during bulk load to be done in parallel using the specified degree of parallelism to be associated with the operation.
- `PARSE` allows parsing of triples retrieved from the staging table (also parses triples containing graph names).
- `<task>_JOIN_HINT=<join_type>`, where <*task*> can be any of the following internal tasks performed during bulk load: `IZC` (is zero collisions), `MBV` (merge batch values), or `MBT` (merge batch triples, used when adding triples to a non-empty model), and where <*join_type*> can be `USE_NL` and `USE_HASH`.

debug
(Reserved for future use)

start_comment
Optional comment about the start of the load operation.

end_comment
Optional comment about the end of the load operation.

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

You must first load semantic data into a staging table before calling this procedure. See Bulk Loading Semantic Data Using a Staging Table for more information.

Using BULK_LOAD_FROM_STAGING_TABLE with Fine Grained Access Control (OLS)

When fine-grained access control (explained in Fine-Grained Access Control for RDF Data) is enabled for the entire network using OLS, only a user with FULL access privileges to the associated policy may perform the bulk load operation. When OLS is enabled, full access privileges to the OLS policy are granted using the SA_USER_ADMIN.SET_USER_PRIVS procedure.

When the OLS is used, the label column in the tables storing the RDF triples must be maintained. By default, with OLS enabled, the label column in the tables storing the RDF triples is set to null. If you have FULL access, you can reset the labels for the newly inserted triples as well as any resources introduced by the new batch of triples by using appropriate subprograms (SEM_RDFSA.SET_RESOURCE_LABEL and SEM_RDFSA.SET_PREDICATE_LABEL).

Optionally, you can define a numeric column named RDF$STC_CTXT1 in the staging table and the application table, to assign the sensitivity label of the triple before the data is loaded into the desired model. Such labels are automatically applied to the corresponding triples stored in the RDF_LINK$ table. The labels for the newly introduced resources may still have to be applied separately before or after the load, and the system does not validate the labels assigned during bulk load operation.

The RDF$STC_CTXT1 column in the application table has no significance, and it may be dropped after the bulk load operation.

By default, SEM_APIS.BULK_LOAD_FROM_STAGING_TABLE uses the semantic network compression setting (stored in RDF_PARAMETER table) for the model.

Examples

The following example loads semantic data stored in the staging table named STAGE_TABLE in schema SCOTT into the semantic model named `family`. The example includes some join hints.

```
EXECUTE SEM_APIS.BULK_LOAD_FROM_STAGING_TABLE('family', 'scott', 'stage_table', flags =>
'IZC_JOIN_HINT=USE_HASH MBV_JOIN_HINT=USE_HASH');
```

15.26 SEM_APIS.BULK_LOAD_RDF_GRAPH

Format

```
SEM_APIS.BULK_LOAD_RDF_GRAPH(
     rdf_graph_name IN VARCHAR2,
     table_owner    IN VARCHAR2,
     table_name     IN VARCHAR2,
     flags          IN VARCHAR2 DEFAULT NULL,
     debug          IN INTEGER DEFAULT NULL,
     start_comment  IN VARCHAR2 DEFAULT NULL,
     end_comment    IN VARCHAR2 DEFAULT NULL,
     network_owner  IN VARCHAR2 DEFAULT NULL,
     network_name   IN VARCHAR2 DEFAULT NULL);
```

Description

Loads semantic data from a staging table.

Parameters

rdf_graph_name
Name of the RDF graph.

table_owner
Name of the schema that owns the staging table that holds semantic data to be loaded.

table_name
Name of the staging table that holds semantic data to be loaded.

flags
An optional quoted string with one or more of the following keyword specifications:

- `COMPRESS=CSCQH` uses COLUMN STORE COMPRESS FOR QUERY HIGH on the RDF_LINK$ partition for the RDF graph.
- `COMPRESS=CSCQL` uses COLUMN STORE COMPRESS FOR QUERY LOW on the RDF_LINK$ partition for the RDF graph.
- COMPRESS=RSCA uses ROW STORE COMPRESS ADVANCED on the RDF_LINK$ partition for the RDF graph.
- COMPRESS=RSCAB uses ROW STORE COMPRESS BASIC on the RDF_LINK$ partition for the RDF graph.
- `DEL_BATCH_DUPS=USE_INSERT` allows the use of an insertion-based strategy for duplicate elimination that may lead to faster processing if the input data contains many duplicates.
- `MBV_METHOD=SHADOW` allows the use of a different value loading strategy that may lead to faster processing for large loads.
- `PARALLEL_CREATE_INDEX` allows internal indexes to be created in parallel, which may improve the performance of the bulk load processing.
- `PARALLEL=<integer>` allows much of the processing used during bulk load to be done in parallel using the specified degree of parallelism to be associated with the operation.

- `PARSE` allows parsing of triples retrieved from the staging table (also parses triples containing graph names).
- `<task>_JOIN_HINT=<join_type>`, where *<task>* can be any of the following internal tasks performed during bulk load: `IZC` (is zero collisions), `MBV` (merge batch values), or `MBT` (merge batch triples, used when adding triples to a non-empty RDF graph), and where *<join_type>* can be `USE_NL` and `USE_HASH`.

debug
(Reserved for future use)

start_comment
Optional comment about the start of the load operation.

end_comment
Optional comment about the end of the load operation.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

You must first load semantic data into a staging table before calling this procedure. See Bulk Loading Semantic Data Using a Staging Table for more information.

Using BULK_LOAD_RDF_GRAPH with Fine Grained Access Control (OLS)

When fine-grained access control (explained in Fine-Grained Access Control for RDF Data) is enabled for the entire network using OLS, only a user with FULL access privileges to the associated policy may perform the bulk load operation. When OLS is enabled, full access privileges to the OLS policy are granted using the SA_USER_ADMIN.SET_USER_PRIVS procedure.

When the OLS is used, the label column in the tables storing the RDF triples must be maintained. By default, with OLS enabled, the label column in the tables storing the RDF triples is set to null. If you have FULL access, you can reset the labels for the newly inserted triples as well as any resources introduced by the new batch of triples by using appropriate subprograms (SEM_RDFSA.SET_RESOURCE_LABEL and SEM_RDFSA.SET_PREDICATE_LABEL).

Optionally, you can define a numeric column named RDF$STC_CTXT1 in the staging table and the application table, to assign the sensitivity label of the triple before the data is loaded into the desired RDF graph. Such labels are automatically applied to the corresponding triples stored in the RDF_LINK$ table. The labels for the newly introduced resources may still have to be applied separately before or after the load, and the system does not validate the labels assigned during bulk load operation.

The RDF$STC_CTXT1 column in the application table has no significance, and it may be dropped after the bulk load operation.

By default, SEM_APIS.BULK_LOAD_RDF_GRAPH uses the RDF network compression setting (stored in RDF_PARAMETER table) for the RDF graph.

Examples

The following example loads semantic data stored in the staging table named STAGE_TABLE in schema SCOTT into the RDF graph named `family`. The example includes some join hints.

```
EXECUTE SEM_APIS.BULK_LOAD_RDF_GRAPH('family', 'scott', 'stage_table', flags =>
'IZC_JOIN_HINT=USE_HASH MBV_JOIN_HINT=USE_HASH');
```

15.27 SEM_APIS.CLEANUP_BNODES

Format

```
SEM_APIS.CLEANUP_BNODES(
     model_name      IN VARCHAR2,
     tablespace_name IN VARCHAR2 DEFAULT NULL,
     options         IN VARCHAR2 DEFAULT NULL);
```

Description

Corrects blank node identifiers for blank nodes in a specified model.

Parameters

model_name
Name of the model.

tablespace_name
Name of the tablespace to use for storing intermediate data.

options
String specifying one or more options to influence the behavior of the procedure. See the Usage Notes for available option values.

Usage Notes

See Blank Nodes: Special Considerations for SPARQL Update.

The `options` parameter can contain one or more of the following keywords:

- `APPEND`: Uses the APPEND hint when populating tables during blank node correction.
- `PARALLEL(n)`: Uses *n* as the degree of parallelism during blank node correction.
- `RECOVER_FAILED=T`: Include this option when a previous attempt to correct blank nodes has been interrupted, and transient tables with intermediate data have not been deleted.

Examples

The following example corrects blank node identifiers for the `electronics` semantic model.

```
EXECUTE SEM_APIS.CLEANUP_BNODES('electronics');
```

15.28 SEM_APIS.CLEANUP_FAILED

Format

```
SEM_APIS.CLEANUP_FAILED(
     rdf_object_type  IN VARCHAR2,
     rdf_object_name  IN VARCHAR2),
     options          IN VARCHAR2 DEFAULT NULL,
     network_owner    IN VARCHAR2 DEFAULT NULL,
     network_name     IN VARCHAR2 default NULL);
```

Description

Drops (deletes) a specified rulebase or entailment if it is in a failed state.

Parameters

rdf_object_type
Type of the RDF object: `RULEBASE` for a rulebase or `RULES_INDEX` for an entailment (rules index).

rdf_object_name
Name of the RDF object of type `rdf_object_type`.

options
(Not currently used.)

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

This procedure checks to see if the specified RDF object is in a failed state; and if the object is in a failed state, the procedure deletes the object.

A rulebase or entailment is in a failed state if a system failure occurred during the creation of that object. You can check if a rulebase or entailment is in a failed state by checking to see if the value of the STATUS column is `FAILED` in the SDO_RULEBASE_INFO view (described in Inferencing: Rules and Rulebases) or the SDO_RULES_INDEX_INFO view (described in Inferred Graphs), respectively.

If the rulebase or entailment is not in a failed state, this procedure performs no action and returns a successful status.

An exception is generated if the RDF object is currently being used.

For information about RDF network types and options, see RDF Networks.

Examples

The following example deletes the rulebase named `family_rb` if (and only if) that rulebase is in a failed state.

```
EXECUTE SEM_APIS.CLEANUP_FAILED('RULEBASE', 'family_rb');
```

15.29 SEM_APIS.COMPOSE_RDF_TERM

Format

```
SEM_APIS.COMPOSE_RDF_TERM(
     value_name    IN VARCHAR2,
     value_type    IN VARCHAR2,
     literal_type  IN VARCHAR2,
     language_type IN VARCHAR2
     ) RETURN VARCHAR2;
```

or

```
SEM_APIS.COMPOSE_RDF_TERM(
     value_name    IN VARCHAR2,
     value_type    IN VARCHAR2,
     literal_type  IN VARCHAR2,
     language_type IN VARCHAR2,
     long_value    IN CLOB,
     options       IN VARCHAR2 DEFAULT NULL,
     ) RETURN CLOB;
```

Description

Creates and returns an RDF term using the specified parameters.

Parameters

value_name
Value name. Must match a value in the VALUE_NAME column in the RDF_VALUE$ table (described in Statements) or in the *var* attribute returned from SEM_MATCH table function.

value_type
The type of text information. Must match a value in the VALUE_TYPE column in the RDF_VALUE$ table (described in Statements) or in the *var*$RDFVTYP attribute returned from SEM_MATCH table function.

literal_type
For typed literals, the type information; otherwise, null. Must either be a null value or match a value in the LITERAL_TYPE column in the RDF_VALUE$ table (described in Statements) or in the *var*$RDFLTYP attribute returned from SEM_MATCH table function.

language_type
Language tag. Must match a value in the LANGUAGE_TYPE column in the RDF_VALUE$ table (described in Statements) or in the *var*$RDFLANG attribute returned from SEM_MATCH table function.

long_value
The character string if the length of the lexical value is greater than 4000 bytes. Must match a value in the LONG_VALUE column in the RDF_VALUE$ table (described in Statements) or in the *var*$RDFCLOB attribute returned from SEM_MATCH table function.

options
(Reserved for future use.)

Usage Notes

If you specify an inconsistent combination of values for the parameters, this function returns a null value. If a null value is returned but you believe that the values for the parameters are appropriate (reflecting columns from the same row in the RDF_VALUE$ table or from a SEM_MATCH query for the same variable), contact Oracle Support.

Examples

The following example returns, for each member of the family whose height is known, the RDF term for the height and also just the value portion of the height.

```
SELECT x, SEM_APIS.COMPOSE_RDF_TERM(h, h$RDFVTYP, h$RDFLTYP, h$RDFLANG)
       h_rdf_term, h
  FROM TABLE(SEM_MATCH(
    '{?x :height ?h}',
```

```
    SEM_Models('family'),
    null,
    SEM_ALIASES(SEM_ALIAS('','http://www.example.org/family/')),
    null))
ORDER BY x;
X
--------------------------------------------------------------------------------
H_RDF_TERM
--------------------------------------------------------------------------------
H
--------------------------------------------------------------------------------
http://www.example.org/family/Cathy
"5.8"^^<http://www.w3.org/2001/XMLSchema#decimal>
5.8

http://www.example.org/family/Cindy
"6"^^<http://www.w3.org/2001/XMLSchema#decimal>
6

http://www.example.org/family/Jack
"6"^^<http://www.w3.org/2001/XMLSchema#decimal>
6

http://www.example.org/family/Tom
"5.75"^^<http://www.w3.org/2001/XMLSchema#decimal>
5.75

4 rows selected.
```

The following example returns the RDF terms for a few of the values stored in the RDF_VALUE$ table.

```
SELECT SEM_APIS.COMPOSE_RDF_TERM(value_name, value_type, literal_type,
     language_type)
  FROM RDF_VALUE$ WHERE ROWNUM < 5;

SEM_APIS.COMPOSE_RDF_TERM(VALUE_NAME,VALUE_TYPE,LITERAL_TYPE,LANGUAGE_TYPE)
--------------------------------------------------------------------------------
<http://www.w3.org/1999/02/22-rdf-syntax-ns#object>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#subject>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#Property>
```

15.30 SEM_APIS.CONVERT_TO_GML311_LITERAL

Format

```
SEM_APIS.CONVERT_TO_GML311_LITERAL(
     geom         IN SDO_GEOMETRY,
     options      IN VARCHAR2 default NULL
     )RETURN CLOB;
```

Description

Serializes an SDO_GEOMETRY object into an `ogc:gmlLiteral` value.

Parameters

geom
SDO_GEOMETRY object to be serialized.

options
(Reserved for future use.)

Usage Notes

The procedure SDO_UTIL.TO_GML311GEOMETRY is used internally to create the geometry literal with a certain spatial reference system URI.

For more information about geometry serialization, see SDO_UTIL.TO_GML311GEOMETRY.

Examples

The following example shows the use of this function for a geometry with SRID 8307 The COLA_MARKETS table is the one from the simple example in *Oracle Spatial Developer's Guide*.

```
INSERT INTO cola_markets VALUES(
  10,
  'cola_x',
  SDO_GEOMETRY(
    2003,
    8307, -- SRID
    NULL,
    SDO_ELEM_INFO_ARRAY(1,1003,3),
    SDO_ORDINATE_ARRAY(1,1, 6,13)
  )
);
commit;

SELECT
sem_apis.convert_to_gml311_literal(shape) as gml1
FROM cola_markets;

"<gml:Polygon srsName=\"SDO:8307\" xmlns:gml=\"http://www.opengis.net/gml\"><gml
:exterior><gml:LinearRing><gml:posList srsDimension=\"2\">1.0 1.0 6.0 1.0 6.0 13.0 1.0
13.0 1.0 1.0 </gml:posList></gml:LinearRing></gml:exterior></gml:Polygon>
"^^<http://www.opengis.net/ont/geosparql#gmlLiteral>
```

15.31 SEM_APIS.CONVERT_TO_WKT_LITERAL

Format

```
SEM_APIS.CONVERT_TO_WKT_LITERAL(
     geom         IN SDO_GEOMETRY,
     srid_prefix  IN VARCHAR2 default NULL,
     options      IN VARCHAR2 default NULL,
     network_owner    IN VARCHAR2 DEFAULT NULL,
     network_name     IN VARCHAR2 default NULL
     )RETURN CLOB;
```

Description

Serializes an SDO_GEOMETRY object into an `ogc:wktLiteral` value.

Parameters

geom
SDO_GEOMETRY object to be serialized.

srid_prefix
Spatial reference system URI prefix that should be used in the `ogc:wktLiteral` instead of the default. The resulting SRID URI will be of the form `<srid_prefix/{srid}>`.

options
String specifying options for transformation. Available options are:

- ORACLE_PREFIX=T. Generate SRID URIs of the form `<http://xmlns.oracle.com/rdf/geo/srid/{srid}>`.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

The procedure SDO_UTIL.TO_WKTGEOMETRY is used internally to create the geometry literal with a certain spatial reference system URI.

Standard SRID URIs are used by default (`<http://www.opengis.net/def/crs/EPSG/0/{srid}>` or (`<http://www.opengis.net/def/crs/OGC/1.3/CRS84>>`).

For more information about geometry serialization, see SDO_UTIL.TO_WKTGEOMETRY.

For information about RDF network types and options, see RDF Networks.

Examples

The following example shows three different uses of this function for a geometry with SRID 8307. The COLA_MARKETS table is the one from the simple example in *Oracle Spatial Developer's Guide*.

```
INSERT INTO cola_markets VALUES(
  10,
  'cola_x',
  SDO_GEOMETRY(
    2003,
    8307, -- SRID
    NULL,
    SDO_ELEM_INFO_ARRAY(1,1003,3),
    SDO_ORDINATE_ARRAY(1,1, 6,13)
  )
);
commit;

SELECT
sem_apis.convert_to_wkt_literal(shape) as wkt1,
sem_apis.convert_to_wkt_literal(shape,'http://my.org/') as wkt2,
sem_apis.convert_to_wkt_literal(shape,null,' ORACLE_PREFIX=T ') as wkt3
FROM cola_markets;

"<http://www.opengis.net/def/crs/OGC/1.3/CRS84> POLYGON ((1.0 1.0, 6.0 1.0, 6.0 13.0,
1.0 13.0, 1.0 1.0))"^^<http://www.opengis.net/ont/geosparql#wktLiteral>
"<http://my.org/8307> POLYGON ((1.0 1.0, 6.0 1.0, 6.0 13.0, 1.0 13.0, 1.0
1.0))"^^<http://www.opengis.net/ont/geosparql#wktLiteral>
"<http://xmlns.oracle.com/rdf/geo/srid/8307> POLYGON ((1.0 1.0, 6.0 1.0, 6.0 13.0, 1.0
13.0, 1.0 1.0))"^^<http://www.opengis.net/ont/geosparql#wktLiteral>
```

15.32 SEM_APIS.CREATE_ENTAILMENT

Format

```
SEM_APIS.CREATE_ENTAILMENT(
     index_name_in      IN VARCHAR2,
     models_in          IN SEM_MODELS,
     rulebases_in       IN SEM_RULEBASES,
     passes             IN NUMBER DEFAULT SEM_APIS.REACH_CLOSURE,
     inf_components_in  IN VARCHAR2 DEFAULT NULL,
     options            IN VARCHAR2 DEFAULT NULL,
     delta_in           IN SEM_MODELS DEFAULT NULL,
     label_gen          IN RDFSA_LABELGEN DEFAULT NULL,
     include_named_g    IN SEM_GRAPHS DEFAULT NULL,
     include_default_g  IN SEM_MODELS DEFAULT NULL,
     include_all_g      IN SEM_MODELS DEFAULT NULL,
     inf_ng_name        IN VARCHAR2 DEFAULT NULL,
     inf_ext_user_func_name IN VARCHAR2 DEFAULT NULL,
     ols_ladder_inf_lbl_sec IN VARCHAR2 DEFAULT NULL,
     network_owner     IN VARCHAR2 DEFAULT NULL,
     network_name      IN VARCHAR2 DEFAULT NULL);
```

> **Note:**
>
> This subprogram will be deprecated in a future release. It is recommended that you use the SEM_APIS.CREATE_INFERRED_GRAPH subprogram instead.

Description

Creates an entailment (rules index) that can be used to perform OWL or RDFS inferencing, and optionally use user-defined rules.

Parameters

index_name_in
Name of the entailment to be created.

models_in
One or more model names. Its data type is SEM_MODELS, which has the following definition: `TABLE OF VARCHAR2(25)`

rulebases_in
One or more rulebase names. Its data type is SEM_RULEBASES, which has the following definition: `TABLE OF VARCHAR2(25)`. Rules and rulebases are explained in Inferencing: Rules and Rulebases.

passes
The number of rounds that the inference engine should run. The default value is `SEM_APIS.REACH_CLOSURE`, which means the inference engine will run till a closure is reached. If the number of rounds specified is less than the number of actual rounds needed to reach a closure, the status of the entailment will then be set to `INCOMPLETE`.

inf_components_in
A comma-delimited string of keywords representing inference components, for performing selective or component-based inferencing. If this parameter is null, the default set of inference components is used. See the Usage Notes for more information about inference components.

options
A comma-delimited string of options to control the inference process by overriding the default inference behavior. To enable an option, specify `option-name=T`; to disable an option, you can specify `option-name=F` (the default). The available option-name values are `COL_COMPRESS`, `DEST_MODEL`, `DISTANCE`, `DOP`, `ENTAIL_ANYWAY`, `HASH_PART`, `INC`, `LOCAL_NG_INF`, `OPT_SAMEAS`, `RAW8`, `PROOF`, and `USER_RULES`. See the Usage Notes for explanations of each value.

delta_in
If incremental inference is in effect, specifies one or more models on which to perform incremental inference. Its data type is SEM_MODELS, which has the following definition: `TABLE OF VARCHAR2(25)`
The triples in the first model in `delta_in` are copied to the first model in `models_in`, and the entailment (rules index) in `rules_index_in` is updated; then the triples in the second model (if any) in `delta_in` are copied to the second model (if any) in `models_in`, and the entailment in `rules_index_in` is updated; and so on until all triples are copied and the entailment is updated. (The `delta_in` parameter has no effect if incremental inference is not enabled for the entailment.)

label_gen
An instance of RDFSA_LABELGEN or a subtype of it, defining the logic for generating Oracle Label Security (OLS) labels for inferred triples. What you specify for this parameter depends on whether you use the default label generator or a custom label generator:

- If you use the default label generator, specify one of the following constants: `SEM_RDFSA.LABELGEN_RULE` for Use Rule Label, `SEM_RDFSA.LABELGEN_SUBJECT` for Use Subject Label, `SEM_RDFSA.LABELGEN_PREDICATE` for Use Predicate Label, `SEM_RDFSA.LABELGEN_OBJECT` for Use Object Label, `SEM_RDFSA.LABELGEN_DOMINATING` for Use Dominating Label, `SEM_RDFSA.LABELGEN_ANTECED` for Use Antecedent Labels.
- If you use a custom label generator, specify the custom label generator type.

include_named_g
Causes all triples from the specified named graphs (across all source models) to participate in named graph based global inference (NGGI, explained in Named Graph Based Global Inference (NGGI)). For example, `include_named_g => sem_graphs('<urn:G1>','<urn:G2>')` implies that triples from named graphs `G1` and `G2` will be included in NGGI.
Its data type is SEM_GRAPHS, which has the following definition: `TABLE OF VARCHAR2(4000)`.

include_default_g
Causes all triples with a null graph name in the specified models to participate in named graph based global inference (NGGI, explained in Named Graph Based Global Inference (NGGI)). For example, `include_default_g => sem_models('m1')` causes all triples with a null graph name from model `M1` to be included in NGGI.

include_all_g
Causes all triples, regardless of their graph name values, in the specified models to participate in named graph based global inference (NGGI, explained in Named Graph Based Global Inference (NGGI)). For example, `include_all_g => sem_models('m2')` causes all triples in model `M2` to be included in NGGI.

inf_ng_name
Assigns the specified graph name to all the new triples inferred by the named graph based global inference (NGGI, explained in Named Graph Based Global Inference (NGGI)).

inf_ext_user_func_name
The name of a user-defined inference function, or a comma-delimited list of names of user-defined functions. For information about creating user-defined inference functions, including format requirements and options for certain parameters, see API Support for User-Defined Inferencing. (For information about user-defined inferencing, including examples, see User-Defined Inferencing and Querying.)

ols_ladder_inf_lbl_sec

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

For the `inf_components_in` parameter, you can specify any combination of the following keywords: `SCOH`, `COMPH`, `DISJH`, `SYMMH`, `INVH`, `SPIH`, `MBRH`, `SPOH`, `DOMH`, `RANH`, `EQCH`, `EQPH`, `FPH`, `IFPH`, `DOM`, `RAN`, `SCO`, `DISJ`, `COMP`, `INV`, `SPO`, `FP`, `IFP`, `SYMM`, `TRANS`, `DIF`, `SAM`, `CHAIN`, `HASKEY`, `ONEOF`, `INTERSECT`, `INTERSECTSCOH`, `MBRLST`, `PROPDISJH`, `SKOSAXIOMS`, `SNOMED`, `SVFH`, `THINGH`, `THINGSAM`, `UNION`, `RDFP1`, `RDFP2`, `RDFP3`, `RDFP4`, `RDFP6`, `RDFP7`, `RDFP8AX`, `RDFP8BX`, `RDFP9`, `RDFP10`, `RDFP11`, `RDFP12A`, `RDFP12B`, `RDFP12C`, `RDFP13A`, `RDFP13B`, `RDFP13C`, `RDFP14A`, `RDFP14BX`, `RDFP15`, `RDFP16`, `RDFS2`, `RDFS3`, `RDFS4a`, `RDFS4b`, `RDFS5`, `RDFS6`, `RDFS7`, `RDFS8`, `RDFS9`, `RDFS10`, `RDFS11`, `RDFS12`, `RDFS13`. For an explanation of the meaning of these keywords, see Table 15-1, where the keywords are listed in alphabetical order.

The default set of inference components for the OWLPrime vocabulary includes the following: `SCOH`, `COMPH`, `DISJH`, `SYMMH`, `INVH`, `SPIH`, `SPOH`, `DOMH`, `RANH`, `EQCH`, `EQPH`, `FPH`, `IFPH`, `SAMH`, `DOM`, `RAN`, `SCO`, `DISJ`, `COMP`, `INV`, `SPO`, `FP`, `IFP`, `SYMM`, `TRANS`, `DIF`, `RDFP14A`, `RDFP14BX`, `RDFP15`, `RDFP16`. However, note the following:

- Component `SAM` is not in this default OWLPrime list, because it tends to generate many new triples for some ontologies.
- Effective with Release 11.2, the native OWL inference engine supports the following new inference components: `CHAIN`, `HASKEY`, `INTERSECT`, `INTERSECTSCOH`, `MBRLST`, `ONEOF`, `PROPDISJH`, `SKOSAXIOMS`, `SNOMED`, `SVFH`, `THINGH`, `THINGSAM`, `UNION`. However, for backward compatibility, the OWLPrime rulebase and any existing rulebases do not include these new components by default; instead, to use these new inference components, you must specify them explicitly, and they are included in Table 15-1 The following example creates an OWLPrime entailment for two OWL ontologies named `LUBM` and `UNIV`. Because of the additional inference components specified, this entailment will include the new semantics introduced in those inference components.

```
EXECUTE sem_apis.create_entailment('lubm1000_idx',sem_models('lubm','univ'),
    sem_rulebases('owlprime'), SEM_APIS.REACH_CLOSURE,
    'INTERSECT,INTERSECTSCOH,SVFH,THINGH,THINGSAM,UNION');
```

Table 15-1 Inferencing Keywords for inf_components_in Parameter

Keyword	Explanation
CHAIN	Captures the property chain semantics defined in OWL 2. Only chains of length 2 are supported. By default, this is included in the `SKOSCORE` rulebase. Subproperty chaining is an OWL 2 feature, and for backward compatibility this component is not by default included in the `OWLPrime` rulebase. (For information about property chain handling, see Property Chain Handling.) (New as of Release 11.2.)
COMPH	Performs inference based on owl:complementOf assertions and the interaction of owl:complementOf with other language constructs.
DIF	Generates owl:differentFrom assertions based on the symmetricity of owl:differentFrom.
DISJ	Infers owl:differentFrom relationships at instance level using owl:disjointWith assertions.
DISJH	Performs inference based on owl:disjointWith assertions and their interactions with other language constructs.
DOM	Performs inference based on RDFS2.
DOMH	Performs inference based on rdfs:domain assertions and their interactions with other language constructs.
EQCH	Performs inference that are relevant to owl:equivalentClass.
EQPH	Performs inference that are relevant to owl:equivalentProperty.
FP	Performs instance-level inference using instances of owl:FunctionalProperty.
FPH	Performs inference using instances of owl:FunctionalProperty.
HASKEY	Covers the semantics behind "keys" defined in OWL 2. In OWL 2, a collection of properties can be treated as a key to a class expression. For efficiency, the size of the collection must not exceed 3. (New as of Release 11.2.)
IFP	Performs instance-level inference using instances of owl:InverseFunctionalProperty.
IFPH	Performs inference using instances of owl:InverseFunctionalProperty.
INTERSECT	Handles the core semantics of owl:intersectionOf. For example, if class C is the intersection of classes C1, C2 and C3, then C is a subclass of C1, C2, and C3. In addition, common instances of all C1, C2, and C3 are also instances of C. (New as of Release 11.2.)
INTERSECTSCOH	Handles the fact that an intersection is the maximal common subset. For example, if class C is the intersection of classes C1, C2, and C3, then any common subclass of all C1, C2, and C3 is a subclass of C. (New as of Release 11.2.)
INV	Performs instance-level inference using owl:inverseOf assertions.
INVH	Performs inference based on owl:inverseOf assertions and their interactions with other language constructs.
MBRLST	Captures the semantics that for any resource, every item in the list given as the value of the `skos:memberList` property is also a value of the `skos:member` property. (See S36 in the SKOS detailed specification.) By default, this is included in the `SKOSCORE` rulebase. (New as of Release 11.2.)
ONEOF	Generates classification assertions based on the definition of the enumeration classes. In OWL, class extensions can be enumerated explicitly with the `owl:oneOf` constructor. (New as of Release 11.2.)

Table 15-1 (Cont.) Inferencing Keywords for inf_components_in Parameter

Keyword	Explanation
PROPDISJH	Captures the interaction between `owl:propertyDisjointWith` and `rdfs:subPropertyOf`. By default, this is included in SKOSCORE rulebase. propertyDisjointWith is an OWL 2 feature, and for backward compatibility this component is not by default included in the `OWLPrime` rulebase. (New as of Release 11.2.)
RANH	Performs inference based on `rdfs:range` assertions and their interactions with other language constructs.
RDFP*	(The rules corresponding to components with a prefix of *RDFP* can be found in *Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary*, by H.J. Horst.)
RDFS2, ... RDFS13	RDFS2, RDFS3, RDFS4a, RDFS4b, RDFS5, RDFS6, RDFS7, RDFS8, RDFS9, RDFS10, RDFS11, RDFS12, and RDFS13 are described in Section 7.3 of *RDF Semantics* (`http://www.w3.org/TR/rdf-mt/`). Note that many of the RDFS components are not relevant for OWL inference.
SAM	Performs inference about individuals based on existing assertions for those individuals and owl:sameAs.
SAMH	Infers owl:sameAs assertions using transitivity and symmetricity of owl:sameAs.
SCO	Performs inference based on RDFS9.
SCOH	Generates the subClassOf hierarchy based on existing rdfs:subClassOf assertions. Basically, C1 rdfs:subClassOf C2 and C2 rdfs:subClassOf C3 will infer C1 rdfs:subClassOf C3 based on transitivity. SCOH is also an alias of RDFS11.
SKOSAXIOMS	Captures most of the axioms defined in the SKOS detailed specification. By default, this is included in the `SKOSCORE` rulebase. (New as of Release 11.2.)
SNOMED	Performs inference based on the semantics of the OWL 2 EL profile, which captures the expressiveness of SNOMED CT (Systematized Nomenclature of Medicine - Clinical Terms), which is one of the most expressive and complex medical terminologies. (New as of Release 11.2.)
SPIH	Performs inference based on interactions between rdfs:subPropertyOf and owl:inverseOf assertions.
SPO	Performs inference based on RDFS7.
SPOH	Generates rdfs:subPropertyOf hierarchy based on transitivity of rdfs:subPropertyOf. It is an alias of RDFS5.
SVFH	Handles the following semantics that involves the interaction between owl:someValuesFrom and rdfs:subClassOf. Consider two existential restriction classes C1 and C2 that both use the same restriction property. Assume further that the owl:someValuesFrom constraint class for C1 is a subclass of that for C2. Then C1 can be inferred as a subclass of C2. (New as of Release 11.2.)
SYMM	Performs instance-level inference using instances of owl:SymmetricProperty.
SYMH	Performs inference for properties of type owl:SymmetricProperty.
THINGH	Handles the semantics that any defined OWL class is a subclass of owl:Thing. The consequence of this rule is that instances of all defined OWL classes will become instances of owl:Thing. The size of the inferred graph will very likely be bigger with this component selected. (New as of Release 11.2.)
THINGSAM	Handles the semantics that instances of owl:Thing are equal to (owl:sameAs) themselves. This component is provided for the convenience of some applications. Note that an application does not have to select this inference component to figure out an individual is equal to itself; this kind of information can easily be built in the application logic. (New as of Release 11.2.)

Table 15-1 (Cont.) Inferencing Keywords for inf_components_in Parameter

Keyword	Explanation
TRANS	Calculates transitive closure for instances of owl:TransitiveProperty.
UNION	Captures the core semantics of the `owl:unionOf` construct. Basically, the `union` class is a superclass of all member classes. For backward compatibility, this component is not by default included in the `OWLPrime` rulebase. (New as of Release 11.2.)

To deselect a component, use the component name followed by a minus (-) sign. For example, `SCOH-` deselects inference of the `subClassOf` hierarchy.

For the `options` parameter, you can enable the following options to override the default inferencing behavior:

- `COL_COMPRESS=T` creates temporary, intermediate working tables. This option can reduce the space required for such tables, and can improve the performance of the CREATE_ENTAILMENT operation with large data sets.

  By default `COL_COMPRESS=T` uses the "compress for query level low" setting; however, you can add `CPQH=T` to change to the "compress for query level high" setting.

  > **Note:**
  >
  > You can specify `COL_COMPRESS=T` only on systems that support Hybrid Columnar Compression (HCC). For information about HCC, see *Oracle Database Concepts*.

- `DEST_MODEL=<model_name>` specifies, for incremental inference, the destination model to which the `delta_in` model or models are to be added. The specified destination model must be one of the models specified in the `models_in` parameter.
- `DISTANCE=T` generates ancillary distance information that is useful for semantic operators.
- `DOP=n` specifies the degree of parallelism for parallel inference, which can improve inference performance. For information about parallel inference, see Using Parallel Inference.
- `ENTAIL_ANYWAY=T` forces OWL inferencing to proceed and reuse existing inferred data (entailment) when the entailment has a valid status. By default, SEM_APIS.CREATE_ENTAILMENT quits immediately if there is already a valid entailment for the combination of models and rulebases.
- `HASH_PART=n` creates the specified number of hash partitions for internal working tables. (The number must be a power of 2: 2, 4, 8, 16, 32, and so on.) You may want to specify a value if there are many distinct predicates in the semantic data model. In Oracle internal testing on benchmark ontologies, HASH_PART=32 worked well.
- `INC=T` enables incremental inference for the entailment. For information about incremental inference, see Performing Incremental Inference.
- `LOCAL_NG_INF=T` causes named graph based *local* inference (NGLI) to be used instead of named graph based global inference (NGGI). For information about NGLI, see Named Graph Based Local Inference (NGLI).

- MODEL_PARTITIONS=*n* overrides the default number of subpartitions in a composite partitioned semantic network and creates the specified number (*n*) of subpartitions in the final entailment partition in RDF_LINK$.
- OPT_SAMEAS=T uses consolidated owl:sameAs entailment for the entailment. If you specify this option, you cannot specify PROOF=T. For information about optimizing owl:sameAs inference, see Optimizing owl:sameAs Inference.
- RAW8=T uses RAW8 data types for the auxiliary inference tables. This option can improve entailment performance by up to 30% in some cases.
- PROOF=T generates proof for inferred triples. Do not specify this option unless you need to; it slows inference performance because it causes more data to be generated. If you specify this option, you cannot specify OPT_SAMEAS=T.
- USER_RULES=T causes any user-defined rules to be applied. If you specify this option, you cannot specify PROOF=T or DISTANCE=T, and you must accept the default value for the passes parameter.

For the delta_in parameter, inference performance is best if the value is small compared to the overall size of those models. In a typical scenario, the best results might be achieved when the delta contains fewer than 10,000 triples; however, some tests have shown significant inference performance improvements with deltas as large as 100,000 triples.

For the label_gen parameter, if you want to use the default OLS label generator, specify the appropriate SEM_RDFSA package constant value fromTable 15-2.

Table 15-2 SEM_RDFSA Package Constants for label_gen Parameter

Constant	Description
SEM_RDFSA.LABELGEN_SUBJECT	Label generator that applies the label associated with the inferred triple's subject as the triple's label.
SEM_RDFSA.LABELGEN_PREDICATE	Label generator that applies the label associated with the inferred triple's subject as the triple's label.
SEM_RDFSA.LABELGEN_OBJECT	Label generator that applies the label associated with the inferred triple's subject as the triple's label.
SEM_RDFSA.LABELGEN_RULE	Label generator that applies the label associated with the rule that directly produced the inferred triple as the triple's label. If you specify this option, you must also specify PROOF=T in the options parameter.
SEM_RDFSA.LABELGEN_DOMINATING	Label generator that computes a dominating label of all the available labels for the triple's components (subject, predicate, object, and rule), and applies it as the label for the inferred triple.

Fine-Grained Access Control (OLS) Considerations

When fine-grained access control is enabled for the entire network using OLS, only a user with FULL access privileges to the associated policy may create an entailment. When OLS is enabled, full access privileges to the OLS policy are granted using the SA_USER_ADMIN.SET_USER_PRIVS procedure.

Inferred triples accessed through generated labels might not be same as conceptual triples inferred directly from the user accessible triples and rules. The labels generated using a subset of triple components may be weaker than intended. For example, one of the antecedents for the inferred triple may have a higher label than any of the components of the triple. When the label is generated based on just the triple components, end users with no access to one of the antecedents may still have access to the inferred triple. Even when the antecedents are used for custom label generation, the generated label may be stronger than intended. The inference process is not exhaustive, and information pertaining to any alternate ways of inferring the same triple is not available. So, the label generated using a given set of antecedents may be too strong, because the user with access to all the triples in the alternate path could infer the triple with lower access.

Even when generating a label that dominates all its components and antecedents, the label may not be precise. This is the case when labels considered for dominating relationship have non-overlapping group information. For example, consider two labels `L:C:NY` and `L:C:NH` where L is a level, C is a component and NY and NH are two groups. A simple label that dominates these two labels is `L:C:NY,NH`, and a true supremum for the two labels is `L:C:US`, where US is parent group for both NY and NH. Unfortunately, neither of these two dominating labels is precise for the triple inferred from the triples with first two labels. If `L:C:NY,NH` is used for the inferred triple, a user with membership in either of these groups has access to the inferred triple, whereas the same user does not have access to one of its antecedents. On the other hand, if `L:C:US` is used for the inferred triple, a user with membership in both the groups and not in the US group will not be able to access the inferred triple, whereas that user could infer the triple by directly accessing its components and antecedents.

Because of these unique challenges with inferred triples, extra caution must be taken when choosing or implementing the label generator.

See also the OLS example in the Examples section.

For information about semantic network types and options, see RDF Networks.

> **Note:**
>
> If the `SEM_APIS.CREATE_ENTAILMENT` procedure with OWL2RL reasoning takes a long time to execute , then the create entailment procedure needs to be executed with options as shown for the OWL2RL rulebase example in the Examples section.

Examples

The following example creates an entailment named `OWLTST_IDX` using the OWLPrime rulebase, and it causes proof to be generated for inferred triples.

```
EXECUTE sem_apis.create_entailment('owltst_idx', sem_models('owltst'),
sem_rulebases('OWLPRIME'), SEM_APIS.REACH_CLOSURE, null, 'PROOF=T');
```

The following example assumes an OLS environment. It creates a rulebase with a rule, and it creates an entailment.

```
-- Create an entailment with a rule. --
exec sdo_rdf_inference.create_entailment('contracts_rb');

insert into rdfr_contracts_rb values (
  'projectLedBy', '(?x :drivenBy ?y) (?y :hasVP ?z)',  NULL,
  '(?x :isLedBy ?z)',
  SDO_RDF_Aliases(SDO_RDF_Alias('','http://www.myorg.com/pred/')));

-- Assign sensitivity label for the predicate to be inferred. --
-- Yhe predicate label may be set globally or it can be assign to --
-- the one or the models used to infer the data - e.g: CONTRACTS.
begin
  sem_rdfsa.set_predicate_label(
         model_name    => 'rdf$global',
         predicate     => 'http://www.myorg.com/pred/isLedBy',
```

```
         label_string => 'TS:US_SPCL');
end;
/

-- Create index with a specific label generator. --
begin
  sem_apis.create_entailment(
         index_name_in  => 'contracts_inf',
         models_in      => SDO_RDF_Models('contracts'),
         rulebases_in   => SDO_RDF_Rulebases('contracts_rb'),
         options        => 'USER_RULES=T',
         label_gen      => sem_rdfsa.LABELGEN_PREDICATE);
end;
/

-- Check for any label exceptions and update them accordingly. --
update rdfi_contracts_inf set ctxt1 = 1100 where ctxt1 = -1;

-- The new entailment is now ready for use in SEM_MATCH queries. --
```

The following example shows the steps to overcome long execution time when creating entailments with OWL2RL rulebase.

```
ALTER SESSION SET "_OPTIMIZER_GENERATE_TRANSITIVE_PRED"=FALSE;
EXECUTE SEM_APIS.CREATE_ENTAILMENT
  ('m1_inf',SEM_MODELS('m1'),SEM_RULEBASES('OWL2RL'),NULL,NULL,
   'RAW8=T,DOP=8,HINTS=[rule:SCM-CLS,use_hash(m1),rule:SCM-OP-
DP,use_hash(m1)],PROCSVF=F,PROCAVF=F,PROCSCMHV=F,PROCSVFH=F,PROCAVFH=F,PROCDOM=F,PROCRAN=
F'
  );
```

15.33 SEM_APIS.CREATE_INDEX_ON_RESULT_TAB

Format

```
SEM_APIS.create_index_on_result_tab (
     index_name         IN VARCHAR2,
     query_pattern_type IN NUMBER,
     result_tab_name    IN VARCHAR2,
     rdf_graph_name     IN VARCHAR2,
     key_string         IN VARCHAR2 DEFAULT NULL,
     tablespace_name    IN DBMS_ID DEFAULT NULL,
     degree             IN NUMBER DEFAULT NULL,
     prefix_length      IN NUMBER DEFAULT NULL,
     options            IN VARCHAR2 DEFAULT NULL,
     network_owner      IN DBMS_ID DEFAULT NULL,
     network_name       IN VARCHAR2 DEFAULT NULL);
```

Description

Creates a unique or a nonunique B-tree index on a result table.

Parameters

index_name
Name of the index.

query_pattern_type
Type of the result table.
The value can be one of the following:

- `SEM_APIS.SPM_TYPE_SVP`
- `SEM_APIS.SPM_TYPE_MVP`
- `SEM_APIS.SPM_TYPE_PCN`

result_tab_name
String for use as part of the name of the result table. If the target is an MVP table, then specify the name of the property.

rdf_graph_name
Name of the RDF graph.

key_string
The index key is a sequence whose elements are columns included in the target result table. It uses an ordinal number based on an ordering, starting from `1`, of the properties in the result table structure. The subject (or `START_NODE_ID`) is in the zeroth position. To include the subject (that is, the `START_NODE_ID` column), use `S`.
To include the object or named graph for the `n-th` property, use `nP` or `nG`, respectively. Thus, `2P` and `2G` would refer to the columns storing the object id and named graph id of the second property in the result table, respectively.
If the subject or a property has in-line lexical values, then they are referred using the format `<n><component-code>`, where `n=0` for the subject. Thus, `0VP` and `2VT`, for example, would refer to the `S_VNAME_PREFIX` and `<2nd_property>_VALUE_TYPE` columns in the result table, respectively.

tablespace_name
Destination tablespace for the index.

degree
Degree of parallelism to use.

prefix_length
Number of leading index key columns to be compressed.

options
Reserved for future use.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

Examples

The following example creates the index `name_idx` on the result table `FLHF` defined on the RDF graph `M1`. The `key_string` parameter, '`2P 1P S`', indicates that the key should be the (numeric id) value from the column corresponding to the second property in the table, followed by that from the first property in the table, followed by the subject (that is, the `START_NODE_ID` column). See Example 1-103 for more details.

```
BEGIN
  SEM_APIS.CREATE_INDEX_ON_RESULT_TAB(
       index_name          => 'name_idx'
     , query_pattern_type => SEM_APIS.SPM_TYPE_SVP
     , result_tab_name    => 'FLHF'
     , rdf_graph_name     => 'M1'
     , key_string         => ' 2P 1P S '
     , network_owner      => 'RDFUSER'
     , network_name       => 'NET1'
  );
END;
/
```

15.34 SEM_APIS.CREATE_INDEX_ON_SPM_TAB

Format

```
SEM_APIS.create_index_on_spm_tab (
     index_name          IN VARCHAR2,
     spm_type            IN NUMBER,
     spm_name            IN VARCHAR2,
     model_name          IN VARCHAR2,
     key_string          IN VARCHAR2 DEFAULT NULL,
     tablespace_name     IN DBMS_ID DEFAULT NULL,
     degree              IN NUMBER DEFAULT NULL,
     prefix_length       IN NUMBER DEFAULT NULL,
     options             IN VARCHAR2 DEFAULT NULL,
     network_owner       IN DBMS_ID DEFAULT NULL,
     network_name        IN VARCHAR2 DEFAULT NULL);
```

> **Note:**
>
> This subprogram will be deprecated in a future release. It is recommended that you use the SEM_APIS.CREATE_INDEX_ON_RESULT_TAB subprogram instead.

Description

Creates a unique or a nonunique B-tree index on Subject-Predicate Matrix table.

Parameters

index_name
Name of the index.

spm_type
Type of the SPM table.
The value can be one of the following:

- `SEM_APIS.SPM_TYPE_SVP`
- `SEM_APIS.SPM_TYPE_MVP`
- `SEM_APIS.SPM_TYPE_PCN`

spm_name
String for use as part of the name of the SPM table. If the target is an MVP table, then specify the name of the property.

model_name
Name of the RDF model.

key_string
The index key is a sequence whose elements are columns included in the target SPM table. It uses an ordinal number based on an ordering, starting from `1`, of the properties in the SPM table structure. The subject (or `START_NODE_ID`) is in the zeroth position. To include the subject (that is, the `START_NODE_ID` column), use `S`.
To include the object or named graph for the `n-th` property, use `nP` or `nG`, respectively. Thus, `2P` and `2G` would refer to the columns storing the object id and named graph id of the second property in the SPM table, respectively.
If the subject or a property has in-line lexical values, then they are referred using the format `<n><component-code>`, where `n=0` for the subject. Thus, `0VP` and `2VT`, for example, would refer to the `S_VNAME_PREFIX` and `<2nd_property>_VALUE_TYPE` columns in the SPM table, respectively.

tablespace_name
Destination tablespace for the index.

degree
Degree of parallelism to use.

prefix_length
Number of leading index key columns to be compressed.

options
Reserved for future use.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

Examples

The following example creates the index `name_idx` on the SVP table `FLHF` defined on model `M1`. The `key_string` parameter, '`2P 1P S`', indicates that the key should be the (numeric id) value from the column corresponding to the second property in the table, followed by that from the first property in the table, followed by the subject (that is, the `START_NODE_ID` column). See Example 1-103 for more details.

```
BEGIN
  SEM_APIS.CREATE_INDEX_ON_SPM_TAB(
       index_name.    => 'name_idx'
     , spm_type       => SEM_APIS.SPM_TYPE_SVP
     , spm_name       => 'FLHF'
     , model_name     => 'M1'
     , key_string     => ' 2P 1P S '
     , network_owner => 'RDFUSER'
     , network_name  => 'NET1'
  );
END;
/
```

15.35 SEM_APIS.CREATE_INFERRED_GRAPH

Format

```
SEM_APIS.CREATE_INFERRED_GRAPH(
     inferred_graph_name          IN VARCHAR2,
     rdf_graphs_in                IN SEM_MODELS,
     rulebases_in                 IN SEM_RULEBASES,
     passes                       IN NUMBER DEFAULT SEM_APIS.REACH_CLOSURE,
     inf_components_in            IN VARCHAR2 DEFAULT NULL,
     options                      IN VARCHAR2 DEFAULT NULL,
     delta_in                     IN SEM_MODELS DEFAULT NULL,
     label_gen                    IN RDFSA_LABELGEN DEFAULT NULL,
     include_named_g              IN SEM_GRAPHS DEFAULT NULL,
     include_default_g            IN SEM_MODELS DEFAULT NULL,
     include_all_g                IN SEM_MODELS DEFAULT NULL,
     inf_ng_name                  IN VARCHAR2 DEFAULT NULL,
     inf_ext_user_func_name       IN VARCHAR2 DEFAULT NULL,
     ols_ladder_inf_lbl_sec       IN VARCHAR2 DEFAULT NULL,
     network_owner                IN VARCHAR2 DEFAULT NULL,
     network_name                 IN VARCHAR2 DEFAULT NULL);
```

Description

Creates an inferred graph (rules index) that can be used to perform OWL or RDFS inferencing, and optionally use user-defined rules.

Parameters

inferred_graph_name
Name of the inferred graph to be created.

rdf_graphs_in
One or more RDF graph names. Its data type is SEM_MODELS, which has the following definition: `TABLE OF VARCHAR2(25)`

rulebases_in
One or more rulebase names. Its data type is SEM_RULEBASES, which has the following definition: `TABLE OF VARCHAR2(25)`. Rules and rulebases are explained in Inferencing: Rules and Rulebases.

passes
The number of rounds that the inference engine should run. The default value is `SEM_APIS.REACH_CLOSURE`, which means the inference engine will run till a closure is reached. If the number of rounds specified is less than the number of actual rounds needed to reach a closure, the status of the inferred graph will then be set to `INCOMPLETE`.

inf_components_in
A comma-delimited string of keywords representing inference components, for performing selective or component-based inferencing. If this parameter is null, the default set of inference components is used. See the Usage Notes for more information about inference components.

options
A comma-delimited string of options to control the inference process by overriding the default inference behavior. To enable an option, specify `*option-name*=T`; to disable an option, you can specify `*option-name*=F` (the default). The available option-name values are `COL_COMPRESS`,

DEST_MODEL, DISTANCE, DOP, ENTAIL_ANYWAY, HASH_PART, INC, LOCAL_NG_INF, OPT_SAMEAS, RAW8, PROOF, and USER_RULES. See the Usage Notes for explanations of each value.

delta_in
If incremental inference is in effect, specifies one or more RDF graphs on which to perform incremental inference. Its data type is SEM_MODELS, which has the following definition: TABLE OF VARCHAR2(25)
The triples in the first RDF graph in delta_in are copied to the first RDF graph in rdf_graphs_in, and the inferred graph (rules index) in rules_index_in is updated; then the triples in the second RDF graph (if any) in delta_in are copied to the second RDF graph (if any) in rdf_graphs_in, and the inferred graph in rules_index_in is updated; and so on until all triples are copied and the inferred graph is updated. (The delta_in parameter has no effect if incremental inference is not enabled for the inferred graph.)

label_gen
An instance of RDFSA_LABELGEN or a subtype of it, defining the logic for generating Oracle Label Security (OLS) labels for inferred triples. What you specify for this parameter depends on whether you use the default label generator or a custom label generator:

- If you use the default label generator, specify one of the following constants: SEM_RDFSA.LABELGEN_RULE for Use Rule Label, SEM_RDFSA.LABELGEN_SUBJECT for Use Subject Label, SEM_RDFSA.LABELGEN_PREDICATE for Use Predicate Label, SEM_RDFSA.LABELGEN_OBJECT for Use Object Label, SEM_RDFSA.LABELGEN_DOMINATING for Use Dominating Label, SEM_RDFSA.LABELGEN_ANTECED for Use Antecedent Labels.
- If you use a custom label generator, specify the custom label generator type.

include_named_g
Causes all triples from the specified named graphs (across all source RDF graphs) to participate in named graph based global inference (NGGI, explained in Named Graph Based Global Inference (NGGI)). For example, include_named_g => sem_graphs('<urn:G1>','<urn:G2>') implies that triples from named graphs G1 and G2 will be included in NGGI.
Its data type is RDF_GRAPHS, which has the following definition: TABLE OF VARCHAR2(4000).

include_default_g
Causes all triples with a null graph name in the specified SEM_MODELS to participate in named graph based global inference (NGGI, explained in Named Graph Based Global Inference (NGGI)). For example, include_default_g => sem_models('m1') causes all triples with a null graph name from M1 to be included in NGGI.

include_all_g
Causes all triples, regardless of their graph name values, in the specified models to participate in named graph based global inference (NGGI, explained in Named Graph Based Global Inference (NGGI)). For example, include_all_g => sem_models('m2') causes all triples in M2 to be included in NGGI.

inf_ng_name
Assigns the specified graph name to all the new triples inferred by the named graph based global inference (NGGI, explained in Named Graph Based Global Inference (NGGI)).

inf_ext_user_func_name
The name of a user-defined inference function, or a comma-delimited list of names of user-defined functions. For information about creating user-defined inference functions, including format requirements and options for certain parameters, see API Support for User-Defined

Inferencing. (For information about user-defined inferencing, including examples, see User-Defined Inferencing and Querying.)

ols_ladder_inf_lbl_sec

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

For the `inf_components_in` parameter, you can specify any combination of the following keywords: `SCOH`, `COMPH`, `DISJH`, `SYMMH`, `INVH`, `SPIH`, `MBRH`, `SPOH`, `DOMH`, `RANH`, `EQCH`, `EQPH`, `FPH`, `IFPH`, `DOM`, `RAN`, `SCO`, `DISJ`, `COMP`, `INV`, `SPO`, `FP`, `IFP`, `SYMM`, `TRANS`, `DIF`, `SAM`, `CHAIN`, `HASKEY`, `ONEOF`, `INTERSECT`, `INTERSECTSCOH`, `MBRLST`, `PROPDISJH`, `SKOSAXIOMS`, `SNOMED`, `SVFH`, `THINGH`, `THINGSAM`, `UNION`, `RDFP1`, `RDFP2`, `RDFP3`, `RDFP4`, `RDFP6`, `RDFP7`, `RDFP8AX`, `RDFP8BX`, `RDFP9`, `RDFP10`, `RDFP11`, `RDFP12A`, `RDFP12B`, `RDFP12C`, `RDFP13A`, `RDFP13B`, `RDFP13C`, `RDFP14A`, `RDFP14BX`, `RDFP15`, `RDFP16`, `RDFS2`, `RDFS3`, `RDFS4a`, `RDFS4b`, `RDFS5`, `RDFS6`, `RDFS7`, `RDFS8`, `RDFS9`, `RDFS10`, `RDFS11`, `RDFS12`, `RDFS13`. For an explanation of the meaning of these keywords, see Table 15-1, where the keywords are listed in alphabetical order.

The default set of inference components for the OWLPrime vocabulary includes the following: `SCOH`, `COMPH`, `DISJH`, `SYMMH`, `INVH`, `SPIH`, `SPOH`, `DOMH`, `RANH`, `EQCH`, `EQPH`, `FPH`, `IFPH`, `SAMH`, `DOM`, `RAN`, `SCO`, `DISJ`, `COMP`, `INV`, `SPO`, `FP`, `IFP`, `SYMM`, `TRANS`, `DIF`, `RDFP14A`, `RDFP14BX`, `RDFP15`, `RDFP16`. However, note the following:

- Component `SAM` is not in this default OWLPrime list, because it tends to generate many new triples for some ontologies.
- Effective with Release 11.2, the native OWL inference engine supports the following new inference components: `CHAIN`, `HASKEY`, `INTERSECT`, `INTERSECTSCOH`, `MBRLST`, `ONEOF`, `PROPDISJH`, `SKOSAXIOMS`, `SNOMED`, `SVFH`, `THINGH`, `THINGSAM`, `UNION`. However, for backward compatibility, the OWLPrime rulebase and any existing rulebases do not include these new components by default; instead, to use these new inference components, you must specify them explicitly, and they are included in Table 15-1 The following example creates an OWLPrime inferred graph for two OWL ontologies named `LUBM` and `UNIV`. Because of the additional inference components specified, this inferred graph will include the new semantics introduced in those inference components.

```
EXECUTE sem_apis.create_inferred_rdf_graph('lubm1000_idx',sem_models('lubm','univ'),
    sem_rulebases('owlprime'), SEM_APIS.REACH_CLOSURE,
    'INTERSECT,INTERSECTSCOH,SVFH,THINGH,THINGSAM,UNION');
```

Table 15-3 Inferencing Keywords for inf_components_in Parameter

Keyword	Explanation
CHAIN	Captures the property chain semantics defined in OWL 2. Only chains of length 2 are supported. By default, this is included in the `SKOSCORE` rulebase. Subproperty chaining is an OWL 2 feature, and for backward compatibility this component is not by default included in the `OWLPrime` rulebase. (For information about property chain handling, see Property Chain Handling.) (New as of Release 11.2.)
COMPH	Performs inference based on owl:complementOf assertions and the interaction of owl:complementOf with other language constructs.

Table 15-3 (Cont.) Inferencing Keywords for inf_components_in Parameter

Keyword	Explanation
DIF	Generates owl:differentFrom assertions based on the symmetricity of owl:differentFrom.
DISJ	Infers owl:differentFrom relationships at instance level using owl:disjointWith assertions.
DISJH	Performs inference based on owl:disjointWith assertions and their interactions with other language constructs.
DOM	Performs inference based on RDFS2.
DOMH	Performs inference based on rdfs:domain assertions and their interactions with other language constructs.
EQCH	Performs inference that are relevant to owl:equivalentClass.
EQPH	Performs inference that are relevant to owl:equivalentProperty.
FP	Performs instance-level inference using instances of owl:FunctionalProperty.
FPH	Performs inference using instances of owl:FunctionalProperty.
HASKEY	Covers the semantics behind "keys" defined in OWL 2. In OWL 2, a collection of properties can be treated as a key to a class expression. For efficiency, the size of the collection must not exceed 3. (New as of Release 11.2.)
IFP	Performs instance-level inference using instances of owl:InverseFunctionalProperty.
IFPH	Performs inference using instances of owl:InverseFunctionalProperty.
INTERSECT	Handles the core semantics of owl:intersectionOf. For example, if class C is the intersection of classes C1, C2 and C3, then C is a subclass of C1, C2, and C3. In addition, common instances of all C1, C2, and C3 are also instances of C. (New as of Release 11.2.)
INTERSECTSCOH	Handles the fact that an intersection is the maximal common subset. For example, if class C is the intersection of classes C1, C2, and C3, then any common subclass of all C1, C2, and C3 is a subclass of C. (New as of Release 11.2.)
INV	Performs instance-level inference using owl:inverseOf assertions.
INVH	Performs inference based on owl:inverseOf assertions and their interactions with other language constructs.
MBRLST	Captures the semantics that for any resource, every item in the list given as the value of the `skos:memberList` property is also a value of the `skos:member` property. (See S36 in the SKOS detailed specification.) By default, this is included in the `SKOSCORE` rulebase. (New as of Release 11.2.)
ONEOF	Generates classification assertions based on the definition of the enumeration classes. In OWL, class extensions can be enumerated explicitly with the `owl:oneOf` constructor. (New as of Release 11.2.)
PROPDISJH	Captures the interaction between `owl:propertyDisjointWith` and `rdfs:subPropertyOf`. By default, this is included in SKOSCORE rulebase. propertyDisjointWith is an OWL 2 feature, and for backward compatibility this component is not by default included in the `OWLPrime` rulebase. (New as of Release 11.2.)
RANH	Performs inference based on `rdfs:range` assertions and their interactions with other language constructs.
RDFP*	(The rules corresponding to components with a prefix of *RDFP* can be found in *Completeness, decidability and complexity of entailment for RDF Schema and a semantic extension involving the OWL vocabulary*, by H.J. Horst.)

Table 15-3 (Cont.) Inferencing Keywords for inf_components_in Parameter

Keyword	Explanation
RDFS2, ... RDFS13	RDFS2, RDFS3, RDFS4a, RDFS4b, RDFS5, RDFS6, RDFS7, RDFS8, RDFS9, RDFS10, RDFS11, RDFS12, and RDFS13 are described in Section 7.3 of *RDF Semantics* (`http://www.w3.org/TR/rdf-mt/`). Note that many of the RDFS components are not relevant for OWL inference.
SAM	Performs inference about individuals based on existing assertions for those individuals and owl:sameAs.
SAMH	Infers owl:sameAs assertions using transitivity and symmetricity of owl:sameAs.
SCO	Performs inference based on RDFS9.
SCOH	Generates the subClassOf hierarchy based on existing rdfs:subClassOf assertions. Basically, C1 rdfs:subClassOf C2 and C2 rdfs:subClassOf C3 will infer C1 rdfs:subClassOf C3 based on transitivity. SCOH is also an alias of RDFS11.
SKOSAXIOMS	Captures most of the axioms defined in the SKOS detailed specification. By default, this is included in the `SKOSCORE` rulebase. (New as of Release 11.2.)
SNOMED	Performs inference based on the semantics of the OWL 2 EL profile, which captures the expressiveness of SNOMED CT (Systematized Nomenclature of Medicine - Clinical Terms), which is one of the most expressive and complex medical terminologies. (New as of Release 11.2.)
SPIH	Performs inference based on interactions between rdfs:subPropertyOf and owl:inverseOf assertions.
SPO	Performs inference based on RDFS7.
SPOH	Generates rdfs:subPropertyOf hierarchy based on transitivity of rdfs:subPropertyOf. It is an alias of RDFS5.
SVFH	Handles the following semantics that involves the interaction between owl:someValuesFrom and rdfs:subClassOf. Consider two existential restriction classes C1 and C2 that both use the same restriction property. Assume further that the owl:someValuesFrom constraint class for C1 is a subclass of that for C2. Then C1 can be inferred as a subclass of C2. (New as of Release 11.2.)
SYMM	Performs instance-level inference using instances of owl:SymmetricProperty.
SYMH	Performs inference for properties of type owl:SymmetricProperty.
THINGH	Handles the semantics that any defined OWL class is a subclass of owl:Thing. The consequence of this rule is that instances of all defined OWL classes will become instances of owl:Thing. The size of the inferred graph will very likely be bigger with this component selected. (New as of Release 11.2.)
THINGSAM	Handles the semantics that instances of owl:Thing are equal to (owl:sameAs) themselves. This component is provided for the convenience of some applications. Note that an application does not have to select this inference component to figure out an individual is equal to itself; this kind of information can easily be built in the application logic. (New as of Release 11.2.)
TRANS	Calculates transitive closure for instances of owl:TransitiveProperty.
UNION	Captures the core semantics of the `owl:unionOf` construct. Basically, the `union` class is a superclass of all member classes. For backward compatibility, this component is not by default included in the `OWLPrime` rulebase. (New as of Release 11.2.)

To deselect a component, use the component name followed by a minus (-) sign. For example, `SCOH-` deselects inference of the `subClassOf` hierarchy.

For the `options` parameter, you can enable the following options to override the default inferencing behavior:

- `COL_COMPRESS=T` creates temporary, intermediate working tables. This option can reduce the space required for such tables, and can improve the performance of the CREATE_INFERRED_GRAPH operation with large data sets.

  By default `COL_COMPRESS=T` uses the "compress for query level low" setting; however, you can add `CPQH=T` to change to the "compress for query level high" setting.

  > **Note:**
  >
  > You can specify `COL_COMPRESS=T` only on systems that support Hybrid Columnar Compression (HCC). For information about HCC, see *Oracle Database Concepts*.

- `DEST_MODEL=<rdf_graph_name>` specifies, for incremental inference, the destination graph to which the `delta_in` RDF graphs are to be added. The specified destination graph must be one of the graphs specified in the `rdf_graphs_in` parameter.
- `DISTANCE=T` generates ancillary distance information that is useful for semantic operators.
- `DOP=n` specifies the degree of parallelism for parallel inference, which can improve inference performance. For information about parallel inference, see Using Parallel Inference.
- `ENTAIL_ANYWAY=T` forces OWL inferencing to proceed and reuse existing inferred data (inferred graph) when the inferred graph has a valid status. By default, SEM_APIS.CREATE_INFERRED_GRAPH quits immediately if there is already a valid inferred graph for the combination of RDF graphs and rulebases.
- `HASH_PART=n` creates the specified number of hash partitions for internal working tables. (The number must be a power of 2: 2, 4, 8, 16, 32, and so on.) You may want to specify a value if there are many distinct predicates in the RDF graph. In Oracle internal testing on benchmark ontologies, HASH_PART=32 worked well.
- `INC=T` enables incremental inference for the inferred graph. For information about incremental inference, see Performing Incremental Inference.
- `LOCAL_NG_INF=T` causes named graph based *local* inference (NGLI) to be used instead of named graph based global inference (NGGI). For information about NGLI, see Named Graph Based Local Inference (NGLI).
- `MODEL_PARTITIONS=n` overrides the default number of subpartitions in a composite partitioned RDF network and creates the specified number (*n*) of subpartitions in the final inferred graph partition in RDF_LINK$.
- `OPT_SAMEAS=T` uses consolidated `owl:sameAs` inferred graph for the inferred graph. If you specify this option, you cannot specify `PROOF=T`. For information about optimizing `owl:sameAs` inference, see Optimizing owl:sameAs Inference.
- `RAW8=T` uses RAW8 data types for the auxiliary inference tables. This option can improve inferred graph performance by up to 30% in some cases.
- `PROOF=T` generates proof for inferred triples. Do not specify this option unless you need to; it slows inference performance because it causes more data to be generated. If you specify this option, you cannot specify `OPT_SAMEAS=T`.

- `USER_RULES=T` causes any user-defined rules to be applied. If you specify this option, you cannot specify `PROOF=T` or `DISTANCE=T`, and you must accept the default value for the `passes` parameter.

For the `delta_in` parameter, inference performance is best if the value is small compared to the overall size of those RDF graphs. In a typical scenario, the best results might be achieved when the delta contains fewer than 10,000 triples; however, some tests have shown significant inference performance improvements with deltas as large as 100,000 triples.

For the `label_gen` parameter, if you want to use the default OLS label generator, specify the appropriate SEM_RDFSA package constant value fromTable 15-2.

Table 15-4 SEM_RDFSA Package Constants for label_gen Parameter

Constant	Description
SEM_RDFSA.LABELGEN_SUBJECT	Label generator that applies the label associated with the inferred triple's subject as the triple's label.
SEM_RDFSA.LABELGEN_PREDICATE	Label generator that applies the label associated with the inferred triple's subject as the triple's label.
SEM_RDFSA.LABELGEN_OBJECT	Label generator that applies the label associated with the inferred triple's subject as the triple's label.
SEM_RDFSA.LABELGEN_RULE	Label generator that applies the label associated with the rule that directly produced the inferred triple as the triple's label. If you specify this option, you must also specify `PROOF=T` in the `options` parameter.
SEM_RDFSA.LABELGEN_DOMINATING	Label generator that computes a dominating label of all the available labels for the triple's components (subject, predicate, object, and rule), and applies it as the label for the inferred triple.

Fine-Grained Access Control (OLS) Considerations

When fine-grained access control is enabled for the entire network using OLS, only a user with FULL access privileges to the associated policy may create an inferred graph. When OLS is enabled, full access privileges to the OLS policy are granted using the `SA_USER_ADMIN.SET_USER_PRIVS` procedure.

Inferred triples accessed through generated labels might not be same as conceptual triples inferred directly from the user accessible triples and rules. The labels generated using a subset of triple components may be weaker than intended. For example, one of the antecedents for the inferred triple may have a higher label than any of the components of the triple. When the label is generated based on just the triple components, end users with no access to one of the antecedents may still have access to the inferred triple. Even when the antecedents are used for custom label generation, the generated label may be stronger than intended. The inference process is not exhaustive, and information pertaining to any alternate ways of inferring the same triple is not available. So, the label generated using a given set of antecedents may be too strong, because the user with access to all the triples in the alternate path could infer the triple with lower access.

Even when generating a label that dominates all its components and antecedents, the label may not be precise. This is the case when labels considered for dominating relationship have non-overlapping group information. For example, consider two labels `L:C:NY` and `L:C:NH` where L is a level, C is a component and NY and NH are two groups. A simple label that dominates these two labels is `L:C:NY,NH`, and a true supremum for the two labels is `L:C:US`, where US is parent group for both NY and NH. Unfortunately, neither of these two dominating labels is precise for the triple inferred from the triples with first two labels. If `L:C:NY,NH` is used for the inferred triple, a user with membership in either of these groups has access to the inferred triple, whereas the same user does not have access to one of its antecedents. On the other hand, if `L:C:US` is used for the inferred triple, a user with membership in both the groups and not in the US group will not be able to access the inferred triple, whereas that user could infer the triple by directly accessing its components and antecedents.

Because of these unique challenges with inferred triples, extra caution must be taken when choosing or implementing the label generator.

See also the OLS example in the Examples section.

For information about RDF network types and options, see RDF Networks.

> **Note:**
>
> If the `SEM_APIS.CREATE_INFERRED_GRAPH` procedure with OWL2RL reasoning takes a long time to execute , then the create inferred graph procedure needs to be executed with options as shown for the OWL2RL rulebase example in the Examples section.

Examples

The following example creates an inferred graph named `OWLTST_IDX` using the OWLPrime rulebase, and it causes proof to be generated for inferred triples.

```
EXECUTE sem_apis.create_inferred_graph('owltst_idx', sem_models('owltst'),
sem_rulebases('OWLPRIME'), SEM_APIS.REACH_CLOSURE, null, 'PROOF=T');
```

The following example assumes an OLS environment. It creates a rulebase with a rule, and it creates an inferred graph.

```
-- Create an inferred graph with a rule. --
exec sdo_rdf_inference.create_inferred_graph('contracts_rb');

insert into rdfr_contracts_rb values (
  'projectLedBy', '(?x :drivenBy ?y) (?y :hasVP ?z)',  NULL,
  '(?x :isLedBy ?z)',
  SDO_RDF_Aliases(SDO_RDF_Alias('','http://www.myorg.com/pred/')));

-- Assign sensitivity label for the predicate to be inferred. --
-- The predicate label may be set globally or it can be assign to --
-- the one or the RDF graphs used to infer the data - e.g: CONTRACTS.
begin
  sem_rdfsa.set_predicate_label(
         model_name   => 'rdf$global',
         predicate    => 'http://www.myorg.com/pred/isLedBy',
         label_string => 'TS:US_SPCL');
end;
/

-- Create index with a specific label generator. --
begin
  sem_apis.create_inferred_graph(
         inferred_graph_name      => 'contracts_inf',
         rdf_graphs_in            => sem_models('contracts'),
         rulebases_in             => sem_Rulebases('contracts_rb'),
         options                  => 'USER_RULES=T',
         label_gen                => sem_rdfsa.LABELGEN_PREDICATE);
end;
/

-- Check for any label exceptions and update them accordingly. --
```

```
update rdfi_contracts_inf set ctxt1 = 1100 where ctxt1 = -1;

-- The new inferred graph is now ready for use in SEM_MATCH queries. --
```

The following example shows the steps to overcome long execution time when creating inferred graphs with OWL2RL rulebase.

```
ALTER SESSION SET "_OPTIMIZER_GENERATE_TRANSITIVE_PRED"=FALSE;
EXECUTE SEM_APIS.CREATE_INFERRED_GRAPH
  ('m1_inf',SEM_MODELS('m1'),SEM_RULEBASES('OWL2RL'),NULL,NULL,
   'RAW8=T,DOP=8,HINTS=[rule:SCM-CLS,use_hash(m1),rule:SCM-OP-
DP,use_hash(m1)],PROCSVF=F,PROCAVF=F,PROCSCMHV=F,PROCSVFH=F,PROCAVFH=F,PROCDOM=F,PROCRAN=
F'
  );
```

15.36 SEM_APIS.CREATE_MATERIALIZED_VIEW

Format

```
SEM_APIS.CREATE_MATERIALIZED_VIEW (
  mv_name        IN VARCHAR2,
  model_name     IN VARCHAR2,
  compression    IN BOOLEAN DEFAULT TRUE,
  inmemory       IN BOOLEAN DEFAULT FALSE,
  values_as_vc   IN BOOLEAN DEFAULT FALSE,
  refresh        IN VARCHAR2 DEFAULT 'C',
  pred_list      IN SYS.ODCIVARCHAR2LIST DEFAULT NULL,
  options        IN VARCHAR2 DEFAULT NULL,
  network_owner  IN VARCHAR2 DEFAULT NULL,
  network_name   IN VARCHAR2 DEFAULT NULL,
);
```

Description

Creates a materialized view for an RDF graph stored in Oracle Database.

Parameters

mv_name
Name of the materialized view to create.

model_name
Name of the model on which to create the materialized view.

compression
Specifies whether the materialized view is compressed.

inmemory
Specifies whether the materialized view is created in IMC format.

values_as_vc
Specifies whether the values of G,S,P,O are created as virtual columns.

refresh
The materialized view refresh method.

pred_list
Specifies the predicates list.

options
String specifying any options to use during the create materialized view operation. Supported options are:

- `TABLESPACE= <name>`: materialized view is created in the named tablespace.
- `PARALLEL= <degree>`: materialized view is created with the parallel degree <degree>.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

For conceptual and usage information, see RDF Support for Materialized Join Views.

For information about RDF network types and options, see RDF Networks.

Examples

The following example creates the materialized view MVX for the RDF model M0.

```
EXECUTE SEM_APIS.CREATE_MATERIALIZED_VIEW('MVX', 'M0');
```

The following example creates the materialized view MVX for the RDF virtual model VM0.

```
EXECUTE SEM_APIS.CREATE_MATERIALIZED_VIEW('MVX', 'VM0');
```

The following example creates the materialized view MVY for the RDF model M1 using the following supported options:

```
EXECUTE SEM_APIS.CREATE_MATERIALIZED_VIEW('MVY','M1',options=>'
TABLESPACE=TBS_3 PARALLEL=2 ');
```

The following example creates the materialized view MVX for the RDF model M0 using a list of predicates.

```
EXECUTE SEM_APIS.CREATE_MATERIALIZED_VIEW('MVX','M0',
pred_list=>sys.odcivarchar2list('http://www.w3.org/2002/07/owl#sameAs',
'http://foo-example.com/name/hasSSN'));
```

15.37 SEM_APIS.SEM_APIS.CREATE_MV_BITMAP_INDEX

Format

```
SEM_APIS.CREATE_MV_BITMAP_INDEX (
  mv_name         IN VARCHAR2,
  idx_columns     IN VARCHAR2,
  options         IN VARCHAR2 DEFAULT NULL,
  network_owner   IN VARCHAR2 DEFAULT NULL,
  network_name    IN VARCHAR2 DEFAULT NULL,
);
```

Description

Creates a bitmap index on a materialized join view for an RDF graph stored in Oracle Database.

Parameters

mv_name
Name of the materialized view on which to create the bitmap index.

idx_columns
Name of the columns on which to create the bitmap index.

options
(Reserved for future use.)

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

For more information, see RDF Support for Materialized Join Views.

For information about RDF network types and options, see RDF Networks.

Examples

The following example creates two bitmap indexes on columns T0P and T1O for the materialized view MVX.

```
EXECUTE SEM_APIS.CREATE_MV_BITMAP_INDEX('MVX', 'T0P T1O');
```

The following example creates five bitmap indexes for the materialized view MVX..

```
EXECUTE SEM_APIS.CREATE_MV_BITMAP_INDEX('MVX', 'T0P T1O T0SV T1OV
T1P$RDFVTYP');
```

15.38 SEM_APIS.CREATE_RDF_GRAPH

Format

```
SEM_APIS.CREATE_RDF_GRAPH(
     rdf_graph_name   IN VARCHAR2,
     table_name       IN VARCHAR2,
     column_name      IN VARCHAR2,
     tablespace_name  IN VARCHAR2 DEFAULT NULL,
     options          IN VARCHAR2 DEFAULT NULL,
     network_owner    IN VARCHAR2 DEFAULT NULL,
     network_name     IN VARCHAR2 DEFAULT NULL);
```

Description

Creates an RDF graph.

Parameters

rdf_graph_name
Name of the RDF graph.

table_name
Name of the table to hold references to semantic technology data for this graph.
This parameter must be `NULL` for a schema-private network.

column_name
Name of the column of type SDO_RDF_TRIPLE_S in `table_name`.
This parameter must be `NULL` for a schema-private network.

tablespace_name
Name of the tablespace for the tables and other database objects used by Oracle to support this graph. The default value is the tablespace that was specified in the call to the SEM_APIS.CREATE_RDF_NETWORK procedure.

options
An optional quoted string with one or more of the following graph creation options:

- `COMPRESS=CSCQH` uses COLUMN STORE COMPRESS FOR QUERY HIGH on the RDF_LINK$ partition for the graph.
- `COMPRESS=CSCQL` uses COLUMN STORE COMPRESS FOR QUERY LOW on the RDF_LINK$ partition for the graph.
- `COMPRESS=RSCA` uses ROW STORE COMPRESS ADVANCED on the RDF_LINK$ partition for the graph.
- `COMPRESS=RSCB` uses ROW STORE COMPRESS BASIC on the RDF_LINK$ partition for the graph.
- `MODEL_PARTITIONS=`*n* overrides the default number of subpartitions in a composite partitioned semantic network and creates the specified number (*n*) of subpartitions in the RDF_LINK$ partition for the graph.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

This procedure adds the RDF graph to the SEM_MODEL$ view, which is described in RDF Graphs.

This procedure is the only supported way to create an RDF graph. Do not use SQL INSERT statements with the SEM_MODEL$ view.

To delete a graph, use the SEM_APIS.DROP_RDF_GRAPH procedure.

The options `COMPRESS=CSCQH`, `COMPRESS=CSCQL`, and `COMPRESS=RSCA` should be used only if you have the appropriate licenses.

For information about RDF network types and options, see RDF Networks.

Examples

The following example creates an RDF graph named `articles` in the schema-private network. (This example is an excerpt from Example 1-129 in Example: Journal Article Information.)

```
EXECUTE SEM_APIS.CREATE_RDF_GRAPH('articles', NULL, NULL,
network_owner=>'RDFUSER', network_name=>'NET1');
```

As part of this operation, a new updatable view, RDFUSER.NET1#RDFT_articles, gets created automatically. You can use this view for any SQL DML statements affecting the data. The following example uses the SDO_RDF_TRIPLE_S constructor to insert data into the graph:

```
INSERT INTO RDFUSER.NET1#RDFT_articles VALUES (
  SDO_RDF_TRIPLE_S ('articles','<http://nature.example.com/Article1>',
    '<http://purl.org/dc/elements/1.1/creator>',
    '"Jane Smith"',
    'RDFUSER',
    'NET1'));
```

15.39 SEM_APIS.CREATE_RDF_GRAPH_COLLECTION

Format

```
SEM_APIS.CREATE_RDF_GRAPH_COLLECTION(
     rdf_graph_collection_name   IN VARCHAR2,
     rdf_graphs                  IN SEM_MODELS,
     rulebases                   IN SEM_RULEBASES DEFAULT NULL,
     options                     IN VARCHAR2 DEFAULT NULL,
     inferred_graphs             IN SEM_ENTAILMENTS DEFAULT NULL,
     network_owner               IN VARCHAR2 DEFAULT NULL,
     network_name                IN VARCHAR2 DEFAULT NULL);
```

Description

Creates an RDF graph collection containing the specified RDF graphs and/or inferred graphs. Inferred graphs can be specified in one of the following ways:

- By specifying one or more RDF graphs and one or more rulebases. In this case, an RDF graph collection will be created using the single entailment that corresponds to the exact combination of graphs and rulebases specified. An error is raised if no such entailment exists.
- By specifying zero or more graphs and one or more entailments. In this case, the contents of the graphs and entailments will be combined regardless of their relationship.

The first method ensures a sound and complete dataset, whereas the second method relaxes the sound and complete constraints for more flexibility.

Parameters

rdf_graph_collection_name
Name of the RDF graph collection to be created.

rdf_graphs
One or more RDF graph names. Its data type is SEM_MODELS, which has the following definition: `TABLE OF VARCHAR2(25)`. If this parameter is null, no graphs are included in the RDF graph collection.

rulebases
One or more rulebase names. Its data type is SEM_RULEBASES, which has the following definition: `TABLE OF VARCHAR2(25)`. If this parameter is null, no rulebases are included in the RDF graph collection. Rules and rulebases are explained in Inferencing: Rules and Rulebases.
If you specify this parameter, you cannot also specify the `inferred_graphs` parameter.

options
Options for creation:

- `PXN=T` forces a UNION ALL-based view definition for the RDF graph collection. This is the default for RDF graph collections with 16 or fewer components.
- `PXN=F` forces an IN LIST-based view definition for the RDF graph collection. This is the default for RDF graph collections with more than 16 components.
- `PXN=F INMEMORY=T` (in combination) let you to create an **in-memory** RDF graph collection.

  If you specify `INMEMORY=T` but not `PXN=F`, then the in-memory virtual columns are created, but the performance will suffer. If you do not specify `INMEMORY=T`, the RDF graph collection is not created in-memory. (See also Using In-Memory Virtual Columns with RDF.)
- `REPLACE=T` lets you to replace an RDF graph collection without dropping it. (Using this option is analogous to using CREATE OR REPLACE VIEW with a view.)

inferred_graphs
One or more inferred graph names. Its data type is SEM_ENTAILMENTS, which has the following definition: `TABLE OF VARCHAR2(25)`. If this parameter is null, no inferred graphs are included in the RDF graph collection. Inferred graphs are explained in Using OWL Inferencing.
If you specify this parameter, you cannot also specify the `rulebases` parameter.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

For an explanation of RDF graph collections, including usage information, see RDF Graph Collections.

An inferred graph must exist for each specified combination of RDF graph and rulebase.

To create an RDF graph collection, you must either be (A) the owner of each specified graph and any corresponding inferred graphs, or (B) a user with DBA privileges.

To replace an RDF graph collection, you must be the owner of the graph or a user with DBA privileges.

The option `INMEMORY=T` should be used only if you have the appropriate licenses.

This procedure creates views with names in the following format:

- SEMV_*vm_name*, which corresponds to a UNION ALL of the triples in each graph and inferred graph. This view may contain duplicates.

- SEMU_*vm_name*, which corresponds to a UNION of the triples in each graph and inferred graph. This view will not contain duplicates (thus, the *U* in SEMU indicates *unique*).

To use the example in RDF Graph Collections of an RDF graph collection vm1 created from graphs m1, m2, m3, and with an entailment created for m1, m2 ,and m3 using the OWLPrime rulebase, this procedure will create the following two views (assuming that m1, m2, and m3, and the OWLPRIME entailment have internal model_id values 1, 2, 3, 4):

```
CREATE VIEW RDFUSER.NET1#.SEMV_VM1 AS
  SELECT p_value_id, start_node_id, canon_end_node_id, end_node_id, g_id, model_id
  FROM RDFUSER.NET1#.rdf_link$ partition (MODEL_1)
UNION ALL
  SELECT p_value_id, start_node_id, canon_end_node_id, end_node_id, g_id, model_id
  FROM RDFUSER.NET1#.rdf_link$ partition (MODEL_2)
UNION ALL
  SELECT p_value_id, start_node_id, canon_end_node_id, end_node_id, g_id, model_id
  FROM RDFUSER.NET1#.rdf_link$ partition (MODEL_3)
UNION ALL
  SELECT p_value_id, start_node_id, canon_end_node_id, end_node_id, g_id, model_id
  FROM RDFUSER.NET1#.rdf_link$ partition (MODEL_4);

CREATE VIEW RDFUSER.NET1#.SEMU_VM1 AS
  SELECT p_value_id, start_node_id, canon_end_node_id, MIN(end_node_id) end_node_id,
g_id, MIN(model_id) model_id
  FROM RDFUSER.NET1#.rdf_link$
  WHERE model_id in (1, 2, 3, 4)
  GROUP BY p_value_id, start_node_id, canon_end_node_id, g_id;
```

The user that invokes this procedure will be the owner of the RDF graph collection and will have SELECT WITH GRANT privileges on the SEMU_*vm_name* and SEMV_*vm_name* views. To query the corresponding RDF graph collection, a user must have select privileges on these views.

For information about RDF network types and options, see RDF Networks.

Examples

The following example creates an RDF graph collection named `VM1`.

```
EXECUTE sem_apis.create_rdf_graph_collection('VM1',
                                          sem_models('model_1', 'model_2'),
                                          sem_rulebases('OWLPRIME'),
                                          network_owner=>'RDFUSER',
                                          network_name=>'NET1');
```

The following example creates an RDF graph collection named `VM1` using the relaxed entailment specification.

```
EXECUTE sem_apis.create_rdf_graph_collection('VM1',
                                          rdf_graphs=>sem_models('model_1', 'model_2'),

inferred_graphs=>sem_entailments('entailment1','entailment2'),
                                          network_owner=>'RDFUSER',
                                          network_name=>'NET1');
```

The following example effectively redefines RDF graph collection `VM1` by using the `REPLACE=T` option.

```
EXECUTE sem_apis.create_rdf_graph_collection('VM1',
                                          rdf_graphs=>sem_models('model_1', 'model_2'),

inferred_graphs=>sem_entailments('entailment1'),
```

```
options=>'REPLACE=T',
network_owner=>'RDFUSER',
network_name=>'NET1');
```

15.40 SEM_APIS.CREATE_RDF_NETWORK

Format

```
SEM_APIS.CREATE_RDF_NETWORK(
     tablespace_name  IN VARCHAR2,
     options          IN VARCHAR2 DEFAULT NULL,
     network_owner    IN VARCHAR2 DEFAULT NULL,
     network_name     IN VARCHAR2 DEFAULT NULL);
```

Description

Creates structures for persistent storage of semantic data.

Parameters

tablespace_name
Name of the tablespace to be used for tables created by this procedure. This tablespace will be the default for all the RDF graphs that you create, although you can override the default when you create a graph by specifying the `model_tablespace` parameter in the call to the SEM_APIS.CREATE_RDF_GRAPH procedure.

options
An optional quoted string with one or more of the following network creation options:

- `COMPRESS=CSCQH` uses COLUMN STORE COMPRESS FOR QUERY HIGH on the RDF_LINK$ and RDF_VALUE$ tables.
- `COMPRESS=CSCQL` uses COLUMN STORE COMPRESS FOR QUERY LOW on the RDF_LINK$ and RDF_VALUE$ tables.
- `COMPRESS=RSCA` uses ROW STORE COMPRESS ADVANCED on the RDF_LINK$ and RDF_VALUE$ tables.
- `COMPRESS=RSCB` uses ROW STORE COMPRESS BASIC on the RDF_LINK$ and RDF_VALUE$ tables. This is the default compression level.
- `MODEL_PARTITIONING=BY_HASH_P` uses list-hash composite partitioning to partition RDF_LINK$ by the graph ID and further subpartition each graph by a hash of the predicate ID.
- `MODEL_PARTITIONS=`*n* sets the default number (*n*) of subpartitions to use for each RDF graph. This option is used in conjunction with `MODEL_PARTITIONING=BY_HASH_P`.
- `MODEL_PARTITIONING=BY_LIST_G` uses list-list composite partitioning to partition RDF_LINK$ by RDF graph `ID` and further subpartition each model by graph `ID`. This subpartition is automatically maintained as data is inserted into the graph.
- `NETWORK_MAX_STRING_SIZE=EXTENDED` specifies a maximum VARCHAR size of 32767 bytes for storing RDF values. Values larger than 32767 bytes will be stored as CLOBs.
- `NETWORK_MAX_STRING_SIZE=STANDARD` specifies a maximum VARCHAR size of 4000 bytes for storing RDF values. Values larger than 4000 bytes will be stored as CLOBs. This is the default.

- `NETWORK_STORAGE_FORM=ESC` specifies use of escaped storage form for lexical values in RDF_VALUE$. Unicode characters and special characters will be stored using ASCII escape sequences. (You cannot specify both the escaped and unescaped storage forms.)
- `NETWORK_STORAGE_FORM=UNESC` specifies use of unescaped storage form for lexical values in RDF_VALUE$. Unicode characters and special characters will be stored as single characters. This is the default.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

This procedure creates system tables and other database objects used for semantic technology support.

You should create a tablespace for the semantic technology system tables and specify the tablespace name in the call to this procedure. (You should *not* specify the `SYSTEM` tablespace.) The size needed for the tablespace that you create will depend on the amount of semantic technology data you plan to store.

You must connect to the database as a user with DBA privileges or as the intended network owner in order to call this procedure, and you should call the procedure only once for the database.

To drop these structures for persistent storage of semantic data, you must connect as a user with DBA privileges or as the owner of the schema-private network, and call the SEM_APIS.DROP_RDF_NETWORK procedure.

The options `COMPRESS=CSCQH`, `COMPRESS=CSCQL`, and `COMPRESS=RSCA` should be used only if you have the appropriate licenses.

After the RDF network is created, a row in the RDF_PARAMETER table with NAMESPACE = 'NETWORK' and ATTRIBUTE = 'COMPRESSION' will indicate the type of compression used for the RDF network.

`NETWORK_MAX_STRING_SIZE=EXTENDED` can only be used if your database has extended VARCHAR support enabled (see Extended Data Types).

For information about RDF network types and options, see RDF Networks.

Examples

The following example creates a tablespace for semantic technology system tables and creates structures for persistent storage of semantic data in this tablespace. Advanced compression is used for the RDF network.

```
CREATE TABLESPACE rdf_tblspace
 DATAFILE '/oradata/orcl/rdf_tblspace.dat' SIZE 1024M REUSE
 AUTOEXTEND ON NEXT 256M MAXSIZE UNLIMITED
 SEGMENT SPACE MANAGEMENT AUTO;
. . .
EXECUTE SEM_APIS.CREATE_RDF_NETWORK('rdf_tblspace',
options=>'MODEL_PARTITIONING=BY_HASH_P MODEL_PARTITIONS=16');
```

15.41 SEM_APIS.CREATE_RDFVIEW_GRAPH

Format

```
SEM_APIS.CREATE_RDFVIEW_GRAPH(
     rdf_graph_name      IN VARCHAR2,
     tables              IN SYS.ODCIVarchar2List,
     prefix              IN VARCHAR2 DEFAULT NULL,
     r2rml_table_owner   IN VARCHAR2 DEFAULT NULL,
     r2rml_table_name    IN VARCHAR2 DEFAULT NULL,
     schema_table_owner  IN VARCHAR2 DEFAULT NULL,
     schema_table_name   IN VARCHAR2 DEFAULT NULL,
     options             IN VARCHAR2 DEFAULT NULL,
     r2rml_string        IN CLOB     DEFAULT NULL,
     r2rml_string_fmt    IN VARCHAR2 DEFAULT NULL,
     network_owner       IN VARCHAR2 DEFAULT NULL,
     network_name        IN VARCHAR2 DEFAULT NULL);
```

Description

Creates an RDF view using direct mapping for the specified list of tables or views or using R2RML mapping.

Parameters

rdf_graph_name
Name of the RDF view to be created.

tables
List of tables or views that are the sources of relational data for the RDF view to be created using direct mapping. This parameter must be null if you want to use R2RML mapping.

prefix
Base prefix to be added at the beginning of the URIs in the RDF view.

r2rml_table_owner
For R2ML mapping, this parameter is required and specifies the name of the schema that owns the staging table that holds the R2RML mapping (in N-triple format) to be used for creating the RDF view.
For direct mapping, this parameter is optional and specifies the name of the schema that owns the staging table into which the R2RML mapping (in N-triple format) generated from the direct mapping will be stored.

r2rml_table_name
For R2ML mapping, this parameter is required and specifies the name of the staging table that holds the R2RML mapping (in N-triple format) to be used for creating the RDF view.
For direct mapping, this parameter is optional and specifies the name of the staging table into which the R2RML mapping (in N-triple format) generated from the direct mapping will be stored.

schema_table_owner
Name of the schema that owns the staging table where the RDF schema generated for the RDF view will be stored.

schema_table_name
Name of the staging table where the RDF schema generated for the RDF view will be stored.

options
For direct mapping, you can optionally specify any combination (including none) of the following:

- `CONFORMANCE=T` suppresses some of the information that would otherwise get included by default, including use of database constraint names and schema-qualified table or view names for constructing RDF predicate names.
- `GENERATE_ONLY=T` only generates the R2RML mapping for the specified tables and stores it in the specified `r2rml_table_name`, but the underlying RDF view graph is not created. If you specify this option, the `r2rml_table_name` parameter must not be null.
- `KEY_BASED_REF_PROPERTY=T` uses the foreign key column names to construct the RDF predicate name. If this option is not specified, then the database constraint name is used for constructing the RDF predicate name.

  For direct mapping, RDF predicate names are derived from the corresponding database names; therefore, preserving the name for the foreign key constraint is the default behavior.

  For an example that uses `KEY_BASED_REF_PROPERTY=T` , see Example 10-1 in Creating an RDF View with Direct Mapping.
- `SCALAR_COLUMNS_ONLY=T` generates the R2RML mapping for only the scalar columns in the specified tables or views. Other non-scalar columns in the tables or views are ignored. Without this option, if you attempt to create a direct mapping on a table with user-defined types or LOB columns, an error is raised.

r2rml_string
An R2RML mapping string in Turtle or N-Triple format to be used for creating the RDF view.

r2rml_string_fmt
The format of the R2RML mapping string specified in `r2rml_string`. Possible values are `TURTLE` and `N-TRIPLE`.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

You must grant the `SELECT` and `INSERT` privileges on `r2rml_table_name` and `schema_table_name` to the network owner.

For more information about RDF views, see RDF Views: Relational Data as RDF.

For information about RDF network types and options, see RDF Networks.

Examples

The following example creates an RDF view using direct mapping for tables EMP and DEPT. The prefix used for the URIs is `http://empdb/`.

```
BEGIN
  sem_apis.create_rdfview_graph(
    rdf_graph_name => 'empdb_model_direct',
    tables => sem_models('EMP', 'DEPT'),
    prefix => 'http://empdb/',
    network_owner=>'RDFUSER',
    network_name=>'NET1'
```

```
  );
END;
/
```

The following example creates an RDF view using R2RML mapping as specified by the RDF triples in the staging table SCOTT.R2RTAB.

```
BEGIN
  sem_apis.create_rdfview_graph(
    rdf_graph_name => 'empdb_model_R2RML',
    tables => NULL,
    r2rml_table_owner => 'SCOTT',
    r2rml_table_name => 'R2RTAB',
    network_owner=>'RDFUSER',
    network_name=>'NET1'
  );
END;
/
```

The following example creates an RDF view using an R2RML mapping specified directly as a string.

```
DECLARE
  r2rmlStr CLOB;
BEGIN

  r2rmlStr :=
   '@prefix rr: <http://www.w3.org/ns/r2rml#>. '||
   '@prefix xsd: <http://www.w3.org/2001/XMLSchema#>. '||
   '@prefix ex: <http://example.com/ns#>. '||'

    ex:TriplesMap_Emp
        rr:logicalTable [ rr:tableName "EMP" ];
        rr:subjectMap [
            rr:template "http://data.example.com/employee/{EMPNO}";
            rr:class ex:Employee;
        ];
        rr:predicateObjectMap [
            rr:predicate ex:empNum;
            rr:objectMap [ rr:column "EMPNO" ; rr:datatype xsd:integer ];
        ];
        rr:predicateObjectMap [
            rr:predicate ex:empName;
            rr:objectMap [ rr:column "ENAME" ];
        ];
        rr:predicateObjectMap [
            rr:predicate ex:jobType;
            rr:objectMap [ rr:column "JOB" ];
        ];
        rr:predicateObjectMap [
            rr:predicate ex:worksForDeptNum;
            rr:objectMap [ rr:column "DEPTNO" ; rr:dataType xsd:integer ];
        ].';

  sem_apis.create_rdfview_graph(
    rdf_graph_name => 'empdb_model_R2RML',
    tables => NULL,
    r2rml_string => r2rmlStr,
    r2rml_string_fmt => 'TURTLE',
    network_owner=>'RDFUSER',
    network_name=>'NET1'
  );
```

```
END;
/
```

The following example creates an RDF view using direct mapping as specified by the RDF triples in the tables EMP and DEPT in the schema-private network owned by RDFUSER. It also selects information about employees who work at the Boston location.

```
BEGIN
  sem_apis.create_rdfview_graph(
  rdf_graph_name => 'empdb_model',
  tables => SYS.ODCIVarchar2List('EMP', 'DEPT'),
  prefix => 'http://empdb/',
  options => 'KEY_BASED_REF_PROPERTY=T',
  network_owner=>'RDFUSER',
  network_name=>'NET1'
  );
END;
/

SELECT e.empno FROM emp e, dept d WHERE e.deptno = d.deptno AND d.loc =
'Boston';

SELECT emp
FROM TABLE(SEM_MATCH('{?emp emp:ref-DEPTNO ?dept . ?dept dept:LOC
"Boston"}',SEM_Models('empdb_model'),NULL,SEM_ALIASES(
SEM_ALIAS('dept','http://empdb/RDFUSER.DEPT#'),SEM_ALIAS('emp','http://empdb/
RDFUSER.EMP#')),null,null,null,null,null,'RDF_USER','NET1'));
```

15.42 SEM_APIS.CREATE_RDFVIEW_MODEL

Format

```
SEM_APIS.CREATE_RDFVIEW_MODEL(
     model_name          IN VARCHAR2,
     tables              IN SYS.ODCIVarchar2List,
     prefix              IN VARCHAR2 DEFAULT NULL,
     r2rml_table_owner   IN VARCHAR2 DEFAULT NULL,
     r2rml_table_name    IN VARCHAR2 DEFAULT NULL,
     schema_table_owner  IN VARCHAR2 DEFAULT NULL,
     schema_table_name   IN VARCHAR2 DEFAULT NULL,
     options             IN VARCHAR2 DEFAULT NULL,
     r2rml_string        IN CLOB     DEFAULT NULL,
     r2rml_string_fmt    IN VARCHAR2 DEFAULT NULL,
     network_owner   IN VARCHAR2 DEFAULT NULL,
     network_name    IN VARCHAR2 DEFAULT NULL);
```

Note:

This subprogram will be deprecated in a future release. It is recommended that you use the SEM_APIS.CREATE_RDFVIEW_GRAPH subprogram instead.

Description

Creates an RDF view using direct mapping for the specified list of tables or views or using R2RML mapping.

Parameters

model_name
Name of the RDF view to be created.

tables
List of tables or views that are the sources of relational data for the RDF view to be created using direct mapping. This parameter must be null if you want to use R2RML mapping.

prefix
Base prefix to be added at the beginning of the URIs in the RDF view.

r2rml_table_owner
For R2ML mapping, this parameter is required and specifies the name of the schema that owns the staging table that holds the R2RML mapping (in N-triple format) to be used for creating the RDF view.
For direct mapping, this parameter is optional and specifies the name of the schema that owns the staging table into which the R2RML mapping (in N-triple format) generated from the direct mapping will be stored.

r2rml_table_name
For R2ML mapping, this parameter is required and specifies the name of the staging table that holds the R2RML mapping (in N-triple format) to be used for creating the RDF view.
For direct mapping, this parameter is optional and specifies the name of the staging table into which the R2RML mapping (in N-triple format) generated from the direct mapping will be stored.

schema_table_owner
Name of the schema that owns the staging table where the RDF schema generated for the RDF view will be stored.

schema_table_name
Name of the staging table where the RDF schema generated for the RDF view will be stored.

options
For direct mapping, you can optionally specify any combination (including none) of the following:

- `CONFORMANCE=T` suppresses some of the information that would otherwise get included by default, including use of database constraint names and schema-qualified table or view names for constructing RDF predicate names.
- `GENERATE_ONLY=T` only generates the R2RML mapping for the specified tables and stores it in the specified `r2rml_table_name`, but the underlying RDF view model is not created. If you specify this option, the `r2rml_table_name` parameter must not be null.
- `KEY_BASED_REF_PROPERTY=T` uses the foreign key column names to construct the RDF predicate name. If this option is not specified, then the database constraint name is used for constructing the RDF predicate name.

For direct mapping, RDF predicate names are derived from the corresponding database names; therefore, preserving the name for the foreign key constraint is the default behavior.

For an example that uses `KEY_BASED_REF_PROPERTY=T` , see Example 10-1 in Creating an RDF View with Direct Mapping.

- `SCALAR_COLUMNS_ONLY=T` generates the R2RML mapping for only the scalar columns in the specified tables or views. Other non-scalar columns in the tables or views are ignored. Without this option, if you attempt to create a direct mapping on a table with user-defined types or LOB columns, an error is raised.

r2rml_string
An R2RML mapping string in Turtle or N-Triple format to be used for creating the RDF view.

r2rml_string_fmt
The format of the R2RML mapping string specified in `r2rml_string`. Possible values are `TURTLE` and `N-TRIPLE`.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

You must grant the `SELECT` and `INSERT` privileges on `r2rml_table_name` and `schema_table_name` to the network owner.

For more information about RDF views, see RDF Views: Relational Data as RDF.

For information about RDF network types and options, see RDF Networks.

Examples

The following example creates an RDF view using direct mapping for tables EMP and DEPT. The prefix used for the URIs is `http://empdb/`.

```
BEGIN
  sem_apis.create_rdfview_model(
    model_name => 'empdb_model_direct',
    tables => sem_models('EMP', 'DEPT'),
    prefix => 'http://empdb/',
    network_owner=>'RDFUSER',
    network_name=>'NET1'
  );
END;
/
```

The following example creates an RDF view using R2RML mapping as specified by the RDF triples in the staging table SCOTT.R2RTAB.

```
BEGIN
  sem_apis.create_rdfview_model(
    model_name => 'empdb_model_R2RML',
    tables => NULL,
    r2rml_table_owner => 'SCOTT',
    r2rml_table_name => 'R2RTAB',
    network_owner=>'RDFUSER',
    network_name=>'NET1'
```

```
  );
END;
/
```

The following example creates an RDF view using an R2RML mapping specified directly as a string.

```
DECLARE
  r2rmlStr CLOB;
BEGIN

  r2rmlStr :=
   '@prefix rr: <http://www.w3.org/ns/r2rml#>. '||
   '@prefix xsd: <http://www.w3.org/2001/XMLSchema#>. '||
   '@prefix ex: <http://example.com/ns#>. '||'

    ex:TriplesMap_Emp
        rr:logicalTable [ rr:tableName "EMP" ];
        rr:subjectMap [
            rr:template "http://data.example.com/employee/{EMPNO}";
            rr:class ex:Employee;
        ];
        rr:predicateObjectMap [
            rr:predicate ex:empNum;
            rr:objectMap [ rr:column "EMPNO" ; rr:datatype xsd:integer ];
        ];
        rr:predicateObjectMap [
            rr:predicate ex:empName;
            rr:objectMap [ rr:column "ENAME" ];
        ];
        rr:predicateObjectMap [
            rr:predicate ex:jobType;
            rr:objectMap [ rr:column "JOB" ];
        ];
        rr:predicateObjectMap [
            rr:predicate ex:worksForDeptNum;
            rr:objectMap [ rr:column "DEPTNO" ; rr:dataType xsd:integer ];
        ].';

  sem_apis.create_rdfview_model(
    model_name => 'empdb_model_R2RML',
    tables => NULL,
    r2rml_string => r2rmlStr,
    r2rml_string_fmt => 'TURTLE',
    network_owner=>'RDFUSER',
    network_name=>'NET1'
  );

END;
/
```

The following example creates an RDF view using direct mapping as specified by the RDF triples in the tables EMP and DEPT in the schema-private network owned by RDFUSER. It also selects information about employees who work at the Boston location.

```
BEGIN
  sem_apis.create_rdfview_model(
  model_name => 'empdb_model',
  tables => SYS.ODCIVarchar2List('EMP', 'DEPT'),
  prefix => 'http://empdb/',
  options => 'KEY_BASED_REF_PROPERTY=T',
```

```
  network_owner=>'RDFUSER',
  network_name=>'NET1'
  );
END;
/

SELECT e.empno FROM emp e, dept d WHERE e.deptno = d.deptno AND d.loc =
'Boston';

SELECT emp
FROM TABLE(SEM_MATCH('{?emp emp:ref-DEPTNO ?dept . ?dept dept:LOC
"Boston"}',SEM_Models('empdb_model'),NULL,SEM_ALIASES(
SEM_ALIAS('dept','http://empdb/RDFUSER.DEPT#'),SEM_ALIAS('emp','http://empdb/
RDFUSER.EMP#')),null,null,null,null,null,'RDF_USER','NET1'));
```

15.43 SEM_APIS.CREATE_RULEBASE

Format

```
SEM_APIS.CREATE_RULEBASE(
     rulebase_name   IN VARCHAR2),
     options         IN VARCHAR2 DEFAULT NULL),
     network_owner   IN VARCHAR2 DEFAULT NULL,
     network_name    IN VARCHAR2 DEFAULT NULL);
```

Description

Creates a rulebase.

Parameters

rulebase_name
Name of the rulebase.

options
(Not currently used.)

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

This procedure creates a user-defined rulebase. After creating the rulebase, you can add rules to it. To cause the rules in the rulebase to be applied in a query of RDF data, you can specify the rulebase in the call to the SEM_MATCH table function.

Rules and rulebases are explained in Inferencing: Rules and Rulebases. The SEM_MATCH table function is described in Using the SEM_MATCH Table Function to Query Semantic Data,

For information about RDF network types and options, see RDF Networks.

Examples

The following example creates a rulebase named family_rb. (It is an excerpt from Example 1-130 in Example: Family Information.)

```
EXECUTE SEM_APIS.CREATE_RULEBASE('family_rb');
```

15.44 SEM_APIS.CREATE_SEM_MODEL

Format

```
SEM_APIS.CREATE_SEM_MODEL(
     model_name       IN VARCHAR2,
     table_name       IN VARCHAR2,
     column_name      IN VARCHAR2,
     model_tablespace IN VARCHAR2 DEFAULT NULL,
     options          IN VARCHAR2 DEFAULT NULL,
     network_owner    IN VARCHAR2 DEFAULT NULL,
     network_name     IN VARCHAR2 DEFAULT NULL);
```

> **Note:**
>
> This subprogram will be deprecated in a future release. It is recommended that you use the SEM_APIS.CREATE_RDF_GRAPH subprogram instead.

Description

Creates a semantic technology model.

Parameters

model_name
Name of the model.

table_name
Name of the table to hold references to semantic technology data for this model.
This parameter must be `NULL` for a schema-private network.

column_name
Name of the column of type SDO_RDF_TRIPLE_S in `table_name`.
This parameter must be `NULL` for a schema-private network.

model_tablespace
Name of the tablespace for the tables and other database objects used by Oracle to support this model. The default value is the tablespace that was specified in the call to the SEM_APIS.CREATE_SEM_NETWORK procedure.

options
An optional quoted string with one or more of the following model creation options:

- `COMPRESS=CSCQH` uses COLUMN STORE COMPRESS FOR QUERY HIGH on the RDF_LINK$ partition for the model.
- `COMPRESS=CSCQL` uses COLUMN STORE COMPRESS FOR QUERY LOW on the RDF_LINK$ partition for the model.
- `COMPRESS=RSCA` uses ROW STORE COMPRESS ADVANCED on the RDF_LINK$ partition for the model.

- `COMPRESS=RSCB` uses ROW STORE COMPRESS BASIC on the RDF_LINK$ partition for the model.
- `MODEL_PARTITIONS=`*n* overrides the default number of subpartitions in a composite partitioned semantic network and creates the specified number (*n*) of subpartitions in the RDF_LINK$ partition for the model.

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

This procedure adds the model to the SEM_MODEL$ view, which is described in Metadata for Models.

This procedure is the only supported way to create a model. Do not use SQL INSERT statements with the SEM_MODEL$ view.

To delete a model, use the SEM_APIS.DROP_SEM_MODEL procedure.

The options `COMPRESS=CSCQH`, `COMPRESS=CSCQL`, and `COMPRESS=RSCA` should be used only if you have the appropriate licenses.

For information about semantic network types and options, see RDF Networks.

Examples

The following example creates a semantic technology model named `articles` in the schema-private network. (This example is an excerpt from Example 1-129 in Example: Journal Article Information.)

```
EXECUTE SEM_APIS.CREATE_SEM_MODEL('articles', NULL, NULL,
network_owner=>'RDFUSER', network_name=>'NET1');
```

As part of this operation, a new updatable view, RDFUSER.NET1#RDFT_articles, gets created automatically. You can use this view for any SQL DML statements affecting the data. The following example uses the SDO_RDF_TRIPLE_S constructor to insert data into the model:

```
INSERT INTO RDFUSER.NET1#RDFT_articles VALUES (
  SDO_RDF_TRIPLE_S ('articles','<http://nature.example.com/Article1>',
    '<http://purl.org/dc/elements/1.1/creator>',
    '"Jane Smith"',
    'RDFUSER',
    'NET1'));
```

15.45 SEM_APIS.CREATE_SEM_NETWORK

Format

```
SEM_APIS.CREATE_SEM_NETWORK(
     tablespace_name  IN VARCHAR2,
     options          IN VARCHAR2 DEFAULT NULL,
```

```
    network_owner     IN VARCHAR2 DEFAULT NULL,
    network_name      IN VARCHAR2 DEFAULT NULL);
```

> **Note:**
>
> This subprogram will be deprecated in a future release. It is recommended that you use the SEM_APIS.CREATE_RDF_NETWORK subprogram instead.

Description

Creates structures for persistent storage of semantic data.

Parameters

tablespace_name
Name of the tablespace to be used for tables created by this procedure. This tablespace will be the default for all models that you create, although you can override the default when you create a model by specifying the `model_tablespace` parameter in the call to the SEM_APIS.CREATE_SEM_MODEL procedure.

options
An optional quoted string with one or more of the following network creation options:

- `COMPRESS=CSCQH` uses COLUMN STORE COMPRESS FOR QUERY HIGH on the RDF_LINK$ and RDF_VALUE$ tables.
- `COMPRESS=CSCQL` uses COLUMN STORE COMPRESS FOR QUERY LOW on the RDF_LINK$ and RDF_VALUE$ tables.
- `COMPRESS=RSCA` uses ROW STORE COMPRESS ADVANCED on the RDF_LINK$ and RDF_VALUE$ tables.
- `COMPRESS=RSCB` uses ROW STORE COMPRESS BASIC on the RDF_LINK$ and RDF_VALUE$ tables. This is the default compression level.
- `MODEL_PARTITIONING=BY_HASH_P` uses list-hash composite partitioning to partition RDF_LINK$ by model ID and further subpartition each model by a hash of the predicate ID.
- `MODEL_PARTITIONS=`*n* sets the default number (*n*) of subpartitions to use for each model. This option is used in conjunction with `MODEL_PARTITIONING=BY_HASH_P`.
- `MODEL_PARTITIONING=BY_LIST_G` uses list-list composite partitioning to partition RDF_LINK$ by model `ID` and further subpartition each model by graph `ID`. This subpartition is automatically maintained as data is inserted into the model.
- `NETWORK_MAX_STRING_SIZE=EXTENDED` specifies a maximum VARCHAR size of 32767 bytes for storing RDF values. Values larger than 32767 bytes will be stored as CLOBs.
- `NETWORK_MAX_STRING_SIZE=STANDARD` specifies a maximum VARCHAR size of 4000 bytes for storing RDF values. Values larger than 4000 bytes will be stored as CLOBs. This is the default.
- `NETWORK_STORAGE_FORM=ESC` specifies use of escaped storage form for lexical values in RDF_VALUE$. Unicode characters and special characters will be stored using ASCII escape sequences. (You cannot specify both the escaped and unescaped storage forms.)

- `NETWORK_STORAGE_FORM=UNESC` specifies use of unescaped storage form for lexical values in RDF_VALUE$. Unicode characters and special characters will be stored as single characters. This is the default.

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

This procedure creates system tables and other database objects used for semantic technology support.

You should create a tablespace for the semantic technology system tables and specify the tablespace name in the call to this procedure. (You should *not* specify the `SYSTEM` tablespace.) The size needed for the tablespace that you create will depend on the amount of semantic technology data you plan to store.

You must connect to the database as a user with DBA privileges or as the intended network owner in order to call this procedure, and you should call the procedure only once for the database.

To drop these structures for persistent storage of semantic data, you must connect as a user with DBA privileges or as the owner of the schema-private network, and call the SEM_APIS.DROP_SEM_NETWORK procedure.

The options `COMPRESS=CSCQH`, `COMPRESS=CSCQL`, and `COMPRESS=RSCA` should be used only if you have the appropriate licenses.

After the semantic network is created, a row in the RDF_PARAMETER table with NAMESPACE = 'NETWORK' and ATTRIBUTE = 'COMPRESSION' will indicate the type of compression used for the semantic network.

`NETWORK_MAX_STRING_SIZE=EXTENDED` can only be used if your database has extended VARCHAR support enabled (see Extended Data Types).

For information about semantic network types and options, see RDF Networks.

Examples

The following example creates a tablespace for semantic technology system tables and creates structures for persistent storage of semantic data in this tablespace. Advanced compression is used for the semantic network.

```
CREATE TABLESPACE rdf_tblspace
 DATAFILE '/oradata/orcl/rdf_tblspace.dat' SIZE 1024M REUSE
 AUTOEXTEND ON NEXT 256M MAXSIZE UNLIMITED
 SEGMENT SPACE MANAGEMENT AUTO;
. . .
EXECUTE SEM_APIS.CREATE_SEM_NETWORK('rdf_tblspace',
options=>'MODEL_PARTITIONING=BY_HASH_P MODEL_PARTITIONS=16');
```

15.46 SEM_APIS.CREATE_SEM_SQL

Format

```
SEM_APIS.CREATE_SEM_SQL;
```

Description

Creates SEM_SQL SQL Macro.

Parameters

Usage Notes

Examples

The following example creates SEM_SQL SQL Macro.

```
EXECUTE SEM_APIS.CREATE_SEM_SQL;
```

15.47 SEM_APIS.CREATE_SOURCE_EXTERNAL_TABLE

Format

```
SEM_APIS.CREATE_SOURCE_EXTERNAL_TABLE(
     source_table   IN VARCHAR2,
     def_directory  IN VARCHAR2,
     log_directory  IN VARCHAR2 DEFAULT NULL,
     bad_directory  IN VARCHAR2 DEFAULT NULL,
     log_file       IN VARCHAR2 DEFAULT NULL,
     bad_file       IN VARCHAR2 DEFAULT NULL,
     parallel       IN INTEGER DEFAULT NULL,
     source_table_owner IN VARCHAR2 DEFAULT NULL,
     flags          IN VARCHAR2 DEFAULT NULL);
```

Description

Creates an external table to map an N-Triple or N-Quad format file into a table.

Parameters

source_table
Name of the external table to be created.

def_directory
Database directory where the input files are located. To load from this staging table, you must have READ privilege on this directory.

log_directory
Database directory where the log files will be generated when loading from the external table. If not specified, the value of the `def_directory` parameter is used. When loading from the external table, you must have WRITE privilege on this directory.

bad_directory
Database directory where the bad files will be generated when loading from the external table. If not specified, the value of the `def_directory` parameter is used. When loading from the external table, you must have WRITE privilege on this directory.

log_file
Name of the log file. If not specified, the name will be generated automatically during a load operation.

bad_file
Name of the bad file. If not specified, the name will be generated automatically during a load operation.

parallel
Degree of parallelism to associate with the external table being created.

source_table_owner
Owner for the external table being created. If not specified, the invoker becomes the owner.

flags
(Reserved for future use)

Usage Notes

For more information and an example, see Loading N-Quad Format Data into a Staging Table Using an External Table.

Examples

The following example creates a source external table. (This example is an excerpt from Example 1-109 in Loading N-Quad Format Data into a Staging Table Using an External Table.)

```
BEGIN
  sem_apis.create_source_external_table(
    source_table     => 'stage_table_source'
   ,def_directory    => 'DATA_DIR'
   ,bad_file         => 'CLOBrows.bad'
   );
END;
```

15.48 SEM_APIS.CREATE_SPARQL_UPDATE_TABLES

Format

```
SEM_APIS.CREATE_SPARQL_UPDATE_TABLES();
```

Description

Creates global temporary tables in the caller's schema for use with SPARQL Update operations.

Parameters

None.

Usage Notes

Invoking SEM_APIS.UPDATE_MODEL with `STREAMING=F`, `FORCE_BULK=T`, or `DEL_AS_INS=T` option requires that the following temporary tables exist in the caller's schema: RDF_UPD_DEL$, RDF_UPD_INS$, and RDF_UPD_INS_CLOB$. These tables are created with the following definitions , where `MAX_STRING_SIZE` is the maximum VARCHAR size for the database:

```
  CREATE GLOBAL TEMPORARY TABLE RDF_UPD_DEL$ (
    RDF$STC_GRAPH VARCHAR2(4000),
    RDF$STC_SUB   VARCHAR2(4000),
    RDF$STC_PRED  VARCHAR2(4000),
```

```
    RDF$STC_OBJ   VARCHAR2(MAX_STRING_SIZE),
    RDF$STC_CLOB  CLOB
  ) ON COMMIT PRESERVE ROWS';
  CREATE GLOBAL TEMPORARY TABLE RDF_UPD_INS$ (
    RDF$STC_GRAPH VARCHAR2(4000),
    RDF$STC_SUB   VARCHAR2(4000),
    RDF$STC_PRED  VARCHAR2(4000),
    RDF$STC_OBJ   VARCHAR2(MAX_STRING_SIZE)
  ) ON COMMIT PRESERVE ROWS';
  CREATE GLOBAL TEMPORARY TABLE RDF_UPD_INS_CLOB$ (
    RDF$STC_GRAPH VARCHAR2(4000),
    RDF$STC_SUB   VARCHAR2(4000),
    RDF$STC_PRED  VARCHAR2(4000),
    RDF$STC_OBJ   VARCHAR2(MAX_STRING_SIZE),
    RDF$STC_CLOB  CLOB
  ) ON COMMIT PRESERVE ROWS';
```

If you need to drop these tables, use the SEM_APIS.DROP_SPARQL_UPDATE_TABLES.

For more information, see Support for SPARQL Update Operations on an RDF Graph.

Examples

The following example creates the necessary global temporary tables in the caller's schema for use with SPARQL Update operations.

```
EXECUTE SEM_APIS.CREATE_SPARQL_UPDATE_TABLES;
```

15.49 SEM_APIS.CREATE_VIRTUAL_MODEL

Format

```
SEM_APIS.CREATE_VIRTUAL_MODEL(
     vm_name       IN VARCHAR2,
     models        IN SEM_MODELS,
     rulebases     IN SEM_RULEBASES DEFAULT NULL,
     options       IN VARCHAR2 DEFAULT NULL,
     entailments   IN SEM_ENTAILMENTS DEFAULT NULL,
     network_owner IN VARCHAR2 DEFAULT NULL,
     network_name  IN VARCHAR2 DEFAULT NULL);
```

> **Note:**
>
> This subprogram will be deprecated in a future release. It is recommended that you use the SEM_APIS.CREATE_RDF_GRAPH_COLLECTION subprogram instead.

Description

Creates a virtual model containing the specified semantic models and/or entailments. Entailments can be specified in one of the following ways:

- By specifying one or more models and one or more rulebases. In this case, a virtual model will be created using the single entailment that corresponds to the exact combination of models and rulebases specified. An error is raised if no such entailment exists.

- By specifying zero or more models and one or more entailments. In this case, the contents of the models and entailments will be combined regardless of their relationship.

The first method ensures a sound and complete dataset, whereas the second method relaxes the sound and complete constraints for more flexibility.

Parameters

vm_name
Name of the virtual model to be created.

models
One or more semantic model names. Its data type is SEM_MODELS, which has the following definition: `TABLE OF VARCHAR2(25)`. If this parameter is null, no models are included in the virtual model definition.

rulebases
One or more rulebase names. Its data type is SEM_RULEBASES, which has the following definition: `TABLE OF VARCHAR2(25)`. If this parameter is null, no rulebases are included in the virtual model definition. Rules and rulebases are explained in Inferencing: Rules and Rulebases.
If you specify this parameter, you cannot also specify the `entailments` parameter.

options
Options for creation:

- `PXN=T` forces a UNION ALL-based view definition for the virtual model. This is the default for virtual models with 16 or fewer components.
- `PXN=F` forces an IN LIST-based view definition for the virtual model. This is the default for virtual models with more than 16 components.
- `PXN=F INMEMORY=T` (in combination) let you to create an **in-memory** virtual model.

  If you specify `INMEMORY=T` but not `PXN=F`, then the in-memory virtual columns are created, but the performance will suffer. If you do not specify `INMEMORY=T`, the virtual model is not created in-memory. (See also Using In-Memory Virtual Columns with RDF.)
- `REPLACE=T` lets you to replace a virtual model without dropping it. (Using this option is analogous to using CREATE OR REPLACE VIEW with a view.)

entailments
One or more entailment names. Its data type is SEM_ENTAILMENTS, which has the following definition: `TABLE OF VARCHAR2(25)`. If this parameter is null, no entailments are included in the virtual model definition. Entailments are explained in Using OWL Inferencing.
If you specify this parameter, you cannot also specify the `rulebases` parameter.

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

For an explanation of virtual models, including usage information, see Virtual Models.

An entailment must exist for each specified combination of semantic model and rulebase.

To create a virtual model, you must either be (A) the owner of each specified model and any corresponding entailments, or (B) a user with DBA privileges.

To replace a virtual model, you must be the owner of the virtual model or a user with DBA privileges.

The option `INMEMORY=T` should be used only if you have the appropriate licenses.

This procedure creates views with names in the following format:

- SEMV_*vm_name*, which corresponds to a UNION ALL of the triples in each model and entailment. This view may contain duplicates.
- SEMU_*vm_name*, which corresponds to a UNION of the triples in each model and entailment. This view will not contain duplicates (thus, the *U* in SEMU indicates *unique*).

To use the example in Virtual Models of a virtual model vm1 created from models m1, m2, m3, and with an entailment created for m1, m2 ,and m3 using the OWLPrime rulebase, this procedure will create the following two views (assuming that m1, m2, and m3, and the OWLPRIME entailment have internal model_id values 1, 2, 3, 4):

```
CREATE VIEW RDFUSER.NET1#.SEMV_VM1 AS
  SELECT p_value_id, start_node_id, canon_end_node_id, end_node_id, g_id, model_id
  FROM RDFUSER.NET1#.rdf_link$ partition (MODEL_1)
UNION ALL
  SELECT p_value_id, start_node_id, canon_end_node_id, end_node_id, g_id, model_id
  FROM RDFUSER.NET1#.rdf_link$ partition (MODEL_2)
UNION ALL
  SELECT p_value_id, start_node_id, canon_end_node_id, end_node_id, g_id, model_id
  FROM RDFUSER.NET1#.rdf_link$ partition (MODEL_3)
UNION ALL
  SELECT p_value_id, start_node_id, canon_end_node_id, end_node_id, g_id, model_id
  FROM RDFUSER.NET1#.rdf_link$ partition (MODEL_4);

CREATE VIEW RDFUSER.NET1#.SEMU_VM1 AS
  SELECT p_value_id, start_node_id, canon_end_node_id, MIN(end_node_id) end_node_id,
g_id, MIN(model_id) model_id
  FROM RDFUSER.NET1#.rdf_link$
  WHERE model_id in (1, 2, 3, 4)
  GROUP BY p_value_id, start_node_id, canon_end_node_id, g_id;
```

The user that invokes this procedure will be the owner of the virtual model and will have SELECT WITH GRANT privileges on the SEMU_*vm_name* and SEMV_*vm_name* views. To query the corresponding virtual model, a user must have select privileges on these views.

For information about semantic network types and options, see RDF Networks.

Examples

The following example creates a virtual model named `VM1`.

```
EXECUTE sem_apis.create_virtual_model('VM1',
                                          sem_models('model_1', 'model_2'),
                                          sem_rulebases('OWLPRIME'),
                                          network_owner=>'RDFUSER',
                                          network_name=>'NET1');
```

The following example creates a virtual model named `VM1` using the relaxed entailment specification.

```
EXECUTE sem_apis.create_virtual_model('VM1',
                                          models=>sem_models('model_1', 'model_2'),

entailments=>sem_entailments('entailment1','entailment2'),
                                          network_owner=>'RDFUSER',
                                          network_name=>'NET1');
```

The following example effectively redefines virtual model `VM1` by using the `REPLACE=T` option.

```
EXECUTE sem_apis.create_virtual_model('VM1',
                                          models=>sem_models('model_1', 'model_2'),
                                          entailments=>sem_entailments('entailment1'),
                                          options=>'REPLACE=T',
                                          network_owner=>'RDFUSER',
                                          network_name=>'NET1');
```

15.50 SEM_APIS.DELETE_ENTAILMENT_STATS

Format

```
SEM_APIS.DELETE_ENTAILMENT_STATS (
     entailment_name   IN VARCHAR2,
     cascade_parts     IN BOOLEAN DEFAULT TRUE,
     cascade_columns   IN BOOLEAN DEFAULT TRUE,
     cascade_indexes   IN BOOLEAN DEFAULT TRUE,
     no_invalidate     IN BOOLEAN DEFAULT DBMS_STATS.AUTO_INVALIDATE,
     force             IN BOOLEAN DEFAULT FALSE,
     network_owner     IN VARCHAR2 DEFAULT NULL,
     network_name      IN VARCHAR2 DEFAULT NULL);
```

Description

Deletes statistics for a specified entailment.

Parameters

entailment_name
Name of the entailment.

(other parameters)
See the parameter explanations for the DBMS_STATS.DELETE_TABLE_STATS procedure in *Oracle Database PL/SQL Packages and Types Reference*, although `force` here applies to entailment statistics.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

See the information about the DBMS_STATS package in*Oracle Database PL/SQL Packages and Types Reference*.

See also Managing Statistics for Semantic Models and the RDF network.

For information about RDF network types and options, see RDF Networks.

Examples

The following example deletes statistics for an entailment named `OWLTST_IDX`.

```
EXECUTE SEM_APIS.DELETE_ENTAILMENT_STATS('owltst_idx');
```

15.51 SEM_APIS.DELETE_MODEL_STATS

Format

```
SEM_APIS.DELETE_MODEL_STATS (
     model_name        IN VARCHAR2,
     cascade_parts     IN BOOLEAN DEFAULT TRUE,
     cascade_columns   IN BOOLEAN DEFAULT TRUE,
     cascade_indexes   IN BOOLEAN DEFAULT TRUE,
     no_invalidate     IN BOOLEAN DEFAULT DBMS_STATS.AUTO_INVALIDATE,
     force             IN BOOLEAN DEFAULT FALSE,
     network_owner     IN VARCHAR2 DEFAULT NULL,
     network_name      IN VARCHAR2 DEFAULT NULL);
```

Description

Deletes statistics for a specified model.

Parameters

model_name
Name of the model.

(other parameters)
See the parameter explanations for the DBMS_STATS.DELETE_TABLE_STATS procedure in *Oracle Database PL/SQL Packages and Types Reference*, although `force` here applies to model statistics.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

Only the model owner or a users with DBA privileges can execute this procedure.

See the information about the DBMS_STATS package in*Oracle Database PL/SQL Packages and Types Reference*.

See also Managing Statistics for Semantic Models and the RDF network.

For information about RDF network types and options, see RDF Networks.

Examples

The following example deletes statistics for a model named `FAMILY`.

```
EXECUTE SEM_APIS.DELETE_MODEL_STATS('family');
```

15.52 SEM_APIS.DISABLE_CHANGE_TRACKING

Format

```
SEM_APIS.DISABLE_CHANGE_TRACKING(
     models_in      IN SEM_MODELS,
```

```
    network_owner IN VARCHAR2 DEFAULT NULL,
    network_name  IN VARCHAR2 DEFAULT NULL);
```

Description

Disables change tracking for a specified set of models.

Parameters

models_in
One or more model names. Its data type is SEM_MODELS, which has the following definition: `TABLE OF VARCHAR2(25)`

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

Disabling change tracking on a model automatically disables incremental inference on all entailment that use the model.

To use this procedure, you must be the owner of the specified model, and incremental inference must have been previously enabled.

For an explanation of incremental inference, including usage information, see Performing Incremental Inference.

For information about RDF network types and options, see RDF Networks.

Examples

The following example disables change tracking for the `family` model.

```
EXECUTE sem_apis.disable_change_tracking(sem_models('family'));
```

15.53 SEM_APIS.DISABLE_INC_INFERENCE

Format

```
SEM_APIS.DISABLE_INC_INFERENCE(
     entailment_name IN VARCHAR2,
     network_owner   IN VARCHAR2 DEFAULT NULL,
     network_name    IN VARCHAR2 DEFAULT NULL);
```

Description

Disables incremental inference for a specified entailment (rules index).

Parameters

entailment_name
Name of the entailment for which to disable incremental inference.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

To use this procedure, you must be the owner of the specified entailment, and incremental inference must have been previously enabled by the SEM_APIS.ENABLE_INC_INFERENCE procedure.

Calling this procedure automatically disables change tracking for all models owned by the invoking user that were having changes tracked only because of this particular inference.

For an explanation of incremental inference, including usage information, see Performing Incremental Inference.

For information about RDF network types and options, see RDF Networks.

Examples

The following example enables incremental inference for the entailment named `RDFS_RIX_FAMILY`.

```
EXECUTE sem_apis.disable_inc_inference('rdfs_rix_family');
```

15.54 SEM_APIS.DISABLE_INMEMORY

Format

```
SEM_APIS.DISABLE_INMEMORY(
     network_owner IN VARCHAR2 DEFAULT NULL,
     network_name  IN VARCHAR2 DEFAULT NULL);
```

Description

Disables in-memory population of RDF data in an RDF network.

Parameters

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

To use this procedure, you must have DBA privileges.

See the information in RDF Support for Oracle Database In-Memory.

For information about RDF network types and options, see RDF Networks.

Examples

The following example disables in-memory population of RDF data in the RDF network.

```
EXECUTE SEM_APIS.DISABLE_INMEMORY;
```

15.55 SEM_APIS.DISABLE_INMEMORY_FOR_ENT

Format

```
SEM_APIS.DISABLE_INMEMORY_FOR_ENT(
     entailment_name IN VARCHAR2,
     network_owner   IN VARCHAR2 DEFAULT NULL,
     network_name    IN VARCHAR2 DEFAULT NULL);
```

> **Note:**
>
> This subprogram will be deprecated in a future release. It is recommended that you use the SEM_APIS.DISABLE_INMEMORY_FOR_INF_GRAPH subprogram instead.

Description

Disables in-memory population of RDF data for an entailment in a semantic network.

Parameters

entailment_name
Name of the entailment.

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

To use this procedure, you must have DBA privileges.

See the information in RDF Support for Oracle Database In-Memory.

For information about semantic network types and options, see RDF Networks.

Examples

The following example disables in-memory population of RDF data for entailment RIDX1 in the semantic network named NET1 owned by RDFUSER.

```
EXECUTE SEM_APIS.DISABLE_INMEMORY_FOR_ENT('RIDX1', network_owner=>'RDFUSER',
network_name=>'NET1');
```

15.56 SEM_APIS.DISABLE_INMEMORY_FOR_MODEL

Format

```
SEM_APIS.DISABLE_INMEMORY_FOR_MODEL(
     model_name    IN VARCHAR2,
     network_owner IN VARCHAR2 DEFAULT NULL,
     network_name  IN VARCHAR2 DEFAULT NULL);
```

> **Note:**
>
> This subprogram will be deprecated in a future release. It is recommended that you use the SEM_APIS.DISABLE_INMEMORY_FOR_RDF_GRAPH subprogram instead.

Description

Disables in-memory population of RDF data for a model in a semantic network.

Parameters

model_name
Name of the model.

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

To use this procedure, you must have DBA privileges.

See the information in RDF Support for Oracle Database In-Memory.

For information about semantic network types and options, see RDF Networks.

Examples

The following example disbles in-memory population of RDF data for model M1 in the semantic network named NET1 owned by RDFUSER.

```
EXECUTE SEM_APIS.DISABLE_INMEMORY_FOR_MODEL('M1', network_owner=>'RDFUSER',
network_name=>'NET1');
```

15.57 SEM_APIS.DISABLE_INMEMORY_FOR_INF_GRAPH

Format

```
SEM_APIS.DISABLE_INMEMORY_FOR_INF_GRAPH(
     inferred_graph_name IN VARCHAR2,
     network_owner       IN VARCHAR2 DEFAULT NULL,
     network_name        IN VARCHAR2 DEFAULT NULL);
```

Description

Disables in-memory population of RDF data for an inferred graph in an RDF network.

Parameters

inferred_graph_name
Name of the inferred graph.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

To use this procedure, you must have DBA privileges.

See the information in RDF Support for Oracle Database In-Memory.

For information about RDF network types and options, see RDF Networks.

Examples

The following example disables in-memory population of RDF data for inferred graph RIDX1 in the RDF network named NET1 owned by RDFUSER.

```
EXECUTE SEM_APIS.DISABLE_INMEMORY_FOR_INF_GRAPH('RIDX1', network_owner=>'RDFUSER',
network_name=>'NET1');
```

15.58 SEM_APIS.DISABLE_INMEMORY_FOR_RDF_GRAPH

Format

```
SEM_APIS.DISABLE_INMEMORY_FOR_RDF_GRAPH(
     rdf_graph_name IN VARCHAR2,
     network_owner  IN VARCHAR2 DEFAULT NULL,
     network_name   IN VARCHAR2 DEFAULT NULL);
```

Description

Disables in-memory population of RDF data for a graph in an RDF network.

Parameters

rdf_graph_name
Name of the RDF graph.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

To use this procedure, you must have DBA privileges.

See the information in RDF Support for Oracle Database In-Memory.

For information about RDF network types and options, see RDF Networks.

Examples

The following example disbles in-memory population of RDF data for graph M1 in the RDF network named NET1 owned by RDFUSER.

```
EXECUTE SEM_APIS.DISABLE_INMEMORY_FOR_RDF_GRAPH('M1', network_owner=>'RDFUSER',
network_name=>'NET1');
```

15.59 SEM_APIS.DISABLE_NETWORK_SHARING

Format

```
SEM_APIS.DISABLE_NETWORK_SHARING(
     network_owner IN VARCHAR2,
     network_name  IN VARCHAR2,
     options       IN VARCHAR2 DEFAULT NULL);
```

Description

Disables sharing of an RDF network.

Parameters

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

options
(Reserved for future use)

Usage Notes

To use this procedure, you must have DBA privileges or be the owner of the specified network.

For information about RDF network types and options, see RDF Networks.

Examples

The following example enables sharing of the `mynetwork` schema-private network owned by database user `scott`.

```
EXECUTE SEM_APIS.DISABLE_NETWORK_SHARING('scott', 'mynetwork');
```

15.60 SEM_APIS.DROP_DATATYPE_INDEX

Format

```
SEM_APIS.DROP_DATATYPE_INDEX(
     datatype      IN VARCHAR2,
     force_drop    IN BOOLEAN default FALSE,
     network_owner IN VARCHAR2 DEFAULT NULL,
     network_name  IN VARCHAR2 DEFAULT NULL);
```

Description

Drops (deletes) an existing data type index.

Parameters

datatype
URI of the data type for the index to drop.

force_drop
TRUE forces the index to be dropped if an error occurs during the processing of the statement; FALSE (the default) does not drop the index if an error occurs during the processing of the statement.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

You must have DBA privileges to call this procedure.

For an explanation of data type indexes, see Using Data Type Indexes.

For information about RDF network types and options, see RDF Networks.

Examples

The following example drops the data type index for `xsd:string` typed literals and plain literals.

```
EXECUTE SEM_APIS.DROP_DATATYPE_INDEX('http://www.w3.org/2001/XMLSchema#string');
```

15.61 SEM_APIS.DROP_ENTAILMENT

Format

```
SEM_APIS.DROP_ENTAILMENT(
     index_name_in IN VARCHAR2,
     named_g_in    IN SEM_GRAPHS DEFAULT NULL,
     dop           IN INT DEFAULT 1,
     network_owner IN VARCHAR2 DEFAULT NULL,
     network_name  IN VARCHAR2 DEFAULT NULL);
```

> **Note:**
>
> This subprogram will be deprecated in a future release. It is recommended that you use the SEM_APIS.DROP_INFERRED_GRAPH subprogram instead.

Description

Drops (deletes) an entailment (rules index).

Parameters

index_name_in
Name of the entailment to be deleted.

named_g_in
Causes only the triples with the specified graph names in the entailment to be deleted. A null value (the default) drops the entire entailment.
For example, `named_g_in => sem_graphs('<urn:G1>','<urn:G2>')` drops only the triples in entailment with graph names G1 and G2; the rest of the entailment graph is not dropped.

dop
Degree of parallelism for a parallel execution of triple deletion. Applies only if the `named_g_in` parameter is not null.

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

You can use this procedure to delete an entailment that you created using the SEM_APIS.CREATE_ENTAILMENT procedure.

If you drop only a subset of the entailment with specified named graphs (that is, when `named_g_in` is not null) on an entailment with a `VALID` or `INCOMPLETE` status, then the resulting status of the entailment after the drop is set to `INCOMPLETE`.

For information about semantic network types and options, see RDF Networks.

Examples

The following example deletes a entailment named `OWLTST_IDX`.

```
EXECUTE sem_apis.drop_entailment('owltst_idx');
```

The following example deletes only inferred triples with graph names `G1` and `G2` that belong to the entailment named `OWLNG_IDX`. Any inferred triples in the default graph and other named graphs remain in the entailment.

```
EXECUTE sem_apis.drop_entailment('owlng_idx',sem_graphs('<urn:G1>','<urn:G2>'));
```

15.62 SEM_APIS.DROP_INFERRED_GRAPH

Format

```
SEM_APIS.DROP_INFERRED_GRAPH(
     inferred_graph_name IN VARCHAR2,
     named_g_in          IN SEM_GRAPHS DEFAULT NULL,
     dop                 IN INT DEFAULT 1,
     network_owner       IN VARCHAR2 DEFAULT NULL,
     network_name        IN VARCHAR2 DEFAULT NULL);
```

Description

Drops (deletes) an inferred graph (rules index).

Parameters

inferred_graph_name
Name of the inferred graph to be deleted.

named_g_in
Causes only the triples with the specified graph names in the inferred graph to be deleted. A null value (the default) drops the entire inferred graph.
For example, `named_g_in => sem_graphs('<urn:G1>','<urn:G2>')` drops only the triples in an inferred graph with graph names `G1` and `G2`; the rest of the inferred graph is not dropped.

dop
Degree of parallelism for a parallel execution of triple deletion. Applies only if the `named_g_in` parameter is not null.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

You can use this procedure to delete an inferred graph that you created using the SEM_APIS.CREATE_INFERRED_GRAPH procedure.

If you drop only a subset of the inferred graph with specified named graphs (that is, when `named_g_in` is not null) on an inferred graph with a `VALID` or `INCOMPLETE` status, then the resulting status of the inferred graph after the drop is set to `INCOMPLETE`.

For information about RDF network types and options, see RDF Networks.

Examples

The following example deletes an inferred graph named `OWLTST_IDX`.

```
EXECUTE sem_apis.drop_inferred_graph('owltst_idx');
```

The following example deletes only inferred triples with graph names `G1` and `G2` that belong to the inferred graph named `OWLNG_IDX`. Any inferred triples in the default graph and other named graphs remain in the inferred graph.

```
EXECUTE sem_apis.drop_inferred_graph('owlng_idx',sem_graphs('<urn:G1>','<urn:G2>'));
```

15.63 SEM_APIS.DROP_MATERIALIZED_VIEW

Format

```
SEM_APIS.DROP_MATERIALIZED_VIEW (
  mv_name         IN VARCHAR2,
  options         IN VARCHAR2 DEFAULT NULL,
  network_owner   IN VARCHAR2 DEFAULT NULL,
  network_name    IN VARCHAR2 DEFAULT NULL,
);
```

Description

Drops a materialized join view for an RDF graph stored in Oracle Database.

Parameters

mv_name
Name of the materialized view to drop.

options
(Reserved for future use.)

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

For more information, see RDF Support for Materialized Join Views.

For information about RDF network types and options, see RDF Networks.

Examples

The following example drops the materialized view MVX.

```
EXECUTE SEM_APIS.DROP_MATERIALIZED_VIEW('MVX');
```

15.64 SEM_APIS.DROP_MV_BITMAP_INDEX

Format

```
SEM_APIS.DROP_MV_BITMAP_INDEX (
  mv_name         IN VARCHAR2,
  idx_columns     IN VARCHAR2,
  options         IN VARCHAR2 DEFAULT NULL,
  network_owner   IN VARCHAR2 DEFAULT NULL,
  network_name    IN VARCHAR2 DEFAULT NULL,
);
```

Description

Drops a bitmap index on a materialized join view for an RDF graph stored in Oracle Database.

Parameters

mv_name
Name of the materialized view from which to drop the bitmap index.

idx_columns
Name of the columns on which to drop the bitmap index.

options
(Reserved for future use.)

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

For more information, see RDF Support for Materialized Join Views.

For information about RDF network types and options, see RDF Networks.

Examples

The following example drops two bitmap indexes on columns T1O and T0SV for the materialized view MVX.

```
EXECUTE SEM_APIS.DROP_MV_BITMAP_INDEX('MVX', 'T1O T0SV');
```

15.65 SEM_APIS.DROP_NETWORK_INDEX

Format

```
SEM_APIS.DROP_NETWORK_INDEX(
     index_code    IN VARCHAR2,
     options       IN VARCHAR2 DEFAULT NULL,
     network_owner IN VARCHAR2 DEFAULT NULL,
     network_name  IN VARCHAR2 DEFAULT NULL);
```

Description

Drops an RDF network index on the RDF graphs and inferred graphs of the RDF network.

Parameters

index_code
Index code string. Must match the `index_code` value that was specified in an earlier call to the SEM_APIS.ADD_SEM_INDEX procedure.

options
(Reserved for future use.)

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

For an explanation of RDF network indexes, see Using Semantic Network Indexes.

For information about RDF network types and options, see RDF Networks.

Examples

The following example drops an RDF network index with the index code string `pcsm` on the RDF graphs and inferred graphs of the RDF network.

```
EXECUTE SEM_APIS.DROP_NETWORK_INDEX('pscm');
```

15.66 SEM_APIS.DROP_RDF_GRAPH

Format

```
SEM_APIS.DROP_RDF_GRAPH(
     rdf_graph_name IN VARCHAR2,
     options        IN VARCHAR2 DEFAULT NULL,
```

```
    network_owner  IN VARCHAR2 DEFAULT NULL,
    network_name   IN VARCHAR2 DEFAULT NULL);
```

Description

Drops (deletes) an RDF graph.

Parameters

rdf_graph_name
Name of the RDF graph.

options
(Reserved for future use.)

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

This procedure deletes the RDF graph from the SEM_MODEL$ view, which is described in RDF Graphs.

This procedure is the only supported way to delete an RDF graph. Do not use SQL DELETE statements with the SEM_MODEL$ view.

Only the creator of the RDF graph can delete the graph.

To truncate an RDF graph instead of deleting it, use the SEM_APIS.TRUNCATE_RDF_GRAPH procedure.

For information about RDF network types and options, see RDF Networks.

Examples

The following example drops the RDF graph named `articles`.

```
EXECUTE SEM_APIS.DROP_RDF_GRAPH('articles');
```

15.67 SEM_APIS.DROP_RDF_GRAPH_COLLECTION

Format

```
SEM_APIS.DROP_RDF_GRAPH_COLLECTION(
     rdf_graph_collection_name IN VARCHAR2,
     options                   IN VARCHAR2 DEFAULT NULL,
     network_owner             IN VARCHAR2 DEFAULT NULL,
     network_name              IN VARCHAR2 DEFAULT NULL);
```

Description

Drops (deletes) an RDF graph collection.

Parameters

rdf_graph_collection_name
Name of the RDF graph collection to be deleted.

options
(Reserved for future use.)

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

You can use this procedure to delete an RDF graph collection that you created using the SEM_APIS.CREATE_RDF_GRAPH_COLLECTION procedure. An RDF graph collection is deleted automatically if any of its component graphs, rulebases, or inferred graph are deleted.

To use this procedure, you must be the owner of the specified RDF graph collection.

For an explanation of RDF graph collections, including usage information, see RDF Graph Collections.

For information about RDF network types and options, see RDF Networks.

Examples

The following example deletes an RDF graph collection named `VM1`.

```
EXECUTE sem_apis.drop_rdf_graph_collection('VM1');
```

15.68 SEM_APIS.DROP_RDF_NETWORK

Format

```
SEM_APIS.DROP_RDF_NETWORK(
     cascade       IN BOOLEAN DEFAULT FALSE,
     options       IN VARCHAR2 DEFAULT NULL,
     network_owner IN VARCHAR2 DEFAULT NULL,
     network_name  IN VARCHAR2 DEFAULT NULL);
```

Description

Removes structures used for persistent storage of semantic data.

Parameters

cascade
`TRUE` drops any existing RDF graphs and rulebases, and removes structures used for persistent storage of semantic data; `FALSE` (the default) causes the operation to fail if any graphs or rulebases exist.

options
(Reserved for future use.)

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

To remove structures used for persistent storage of semantic data, you must connect as a user with DBA privileges or as the owner of the schema-private network, and call this procedure.

If any version-enabled RDF graphs exist, this procedure will fail regardless of the value of the `cascade` parameter.

For information about RDF network types and options, see RDF Networks.

Examples

The following example removes structures used for persistent storage of semantic data.

```
EXECUTE SEM_APIS.DROP_RDF_NETWORK;
```

15.69 SEM_APIS.DROP_RDFVIEW_GRAPH

Format

```
SEM_APIS.DROP_RDFVIEW_GRAPH(
     rdf_graph_name IN VARCHAR2,
     options        IN VARCHAR2 DEFAULT NULL,
     network_owner  IN VARCHAR2 DEFAULT NULL,
     network_name   IN VARCHAR2 DEFAULT NULL);
```

Description

Drops (deletes) an RDF view.

Parameters

rdf_graph_name
Name of the RDF view to be dropped.

options
(Reserved for future use.)

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

You must be the owner of the RDF view to be dropped.

For more information about RDF views, see RDF Views: Relational Data as RDF.

For information about RDF network types and options, see RDF Networks.

Examples

The following example drops an RDF view.

```
BEGIN
  sem_apis.drop_rdfview_graph(
    rdf_graph_name => 'empdb_model'
  );
END;
/
```

15.70 SEM_APIS.DROP_RDFVIEW_MODEL

Format

```
SEM_APIS.DROP_RDFVIEW_MODEL(
     model_name    IN VARCHAR2,
     options       IN VARCHAR2 DEFAULT NULL,
     network_owner IN VARCHAR2 DEFAULT NULL,
     network_name  IN VARCHAR2 DEFAULT NULL);
```

> **Note:**
>
> This subprogram will be deprecated in a future release. It is recommended that you use the SEM_APIS.DROP_RDFVIEW_GRAPH subprogram instead.

Description

Drops (deletes) an RDF view.

Parameters

model_name
Name of the RDF view to be dropped.

options
(Reserved for future use.)

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

You must be the owner of the RDF view to be dropped.

For more information about RDF views, see RDF Views: Relational Data as RDF.

For information about RDF network types and options, see RDF Networks.

Examples

The following example drops an RDF view.

```
BEGIN
  sem_apis.drop_rdfview_model(
    model_name => 'empdb_model'
  );
END;
/
```

15.71 SEM_APIS.DROP_RESULT_TAB

Format

```
SEM_APIS.DROP_RESULT_TAB (
     query_pattern_type IN NUMBER,
     result_tab_name    IN VARCHAR2,
     rdf_graph_name     IN VARCHAR2,
     options            IN VARCHAR2 DEFAULT NULL,
     network_owner      IN DBMS_ID DEFAULT NULL,
     network_name       IN VARCHAR2 DEFAULT NULL);
```

Description

Drops result table(s) defined on a given RDF graph.

Parameters

query_pattern_type
Type of the result table.
The value can be one of the following:

- `SEM_APIS.SPM_TYPE_SVP`
- `SEM_APIS.SPM_TYPE_MVP`
- `SEM_APIS.SPM_TYPE_PCN`
- `SEM_APIS.SPM_TYPE_ALL`

  Use of `SEM_APIS.SPM_TYPE_ALL` indicates that the target result tables can be of any type.

result_tab_name
String for use as part of the name of the result table. If the target is an MVP table, then specify the name of the property. Use of the value '`*`' indicates that the target is the set of all the result tables of the type specified by the `spm_type` parameter.

rdf_graph_name
Name of the RDF graph.

options
Reserved for future use.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

- This operation has a DDL semantics.
- The invoker must be the owner of the target RDF graph or the RDF network or both.

Examples

The following example drops a specific SVP table named `FLHF`:

```
BEGIN
  SEM_APIS.DROP_RESULT_TAB(
     query_pattern_type => SEM_APIS.SPM_TYPE_SVP
   , result_tab_name    => 'FLHF'
   , rdf_graph_name     => 'M1'
   , network_owner      => 'RDFUSER'
   , network_name       => 'NET1'
  );
END;
/
```

The following example drops all the currently existing result tables on RDF graph `M1`:

```
BEGIN
  SEM_APIS.DROP_RESULT_TAB(
     query_pattern_type   => SEM_APIS.SPM_TYPE_ALL
   , resulrdf_graph_name  => 'M1'
   , network_owner        => 'RDFUSER'
   , network_name         => 'NET1'
  );
END;
/
```

15.72 SEM_APIS.DROP_RULEBASE

Format

```
SEM_APIS.DROP_RULEBASE(
     rulebase_name   IN VARCHAR2,
     options         IN VARCHAR2 DEFAULT NULL,
     network_owner   IN VARCHAR2 DEFAULT NULL,
     network_name    IN VARCHAR2 DEFAULT NULL);
```

Description

Deletes a rulebase.

Parameters

rulebase_name
Name of the rulebase.

options
(Reserved for future use.)

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

This procedure deletes the specified rulebase, making it no longer available for use in calls to the SEM_MATCH table function. For information about rulebases, see Inferencing: Rules and Rulebases.

Only the creator of a rulebase can delete the rulebase.

For information about RDF network types and options, see RDF Networks.

Examples

The following example drops the rulebase named `family_rb`.

```
EXECUTE SEM_APIS.DROP_RULEBASE('family_rb');
```

15.73 SEM_APIS.DROP_SEM_INDEX

Format

```
SEM_APIS.DROP_SEM_INDEX(
     index_code    IN VARCHAR2,
     options       IN VARCHAR2 DEFAULT NULL,
     network_owner IN VARCHAR2 DEFAULT NULL,
     network_name  IN VARCHAR2 DEFAULT NULL);
```

> **Note:**
>
> This subprogram will be deprecated in a future release. It is recommended that you use the SEM_APIS.DROP_NETWORK_INDEX subprogram instead.

Description

Drops a semantic network index on the models and entailments of the semantic network.

Parameters

index_code
Index code string. Must match the `index_code` value that was specified in an earlier call to the SEM_APIS.ADD_SEM_INDEX procedure.

options
(Reserved for future use.)

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

For an explanation of semantic network indexes, see Using Semantic Network Indexes.

For information about semantic network types and options, see RDF Networks.

Examples

The following example drops a semantic network index with the index code string `pcsm` on the models and entailments of the semantic network.

```
EXECUTE SEM_APIS.DROP_SEM_INDEX('pscm');
```

15.74 SEM_APIS.DROP_SEM_MODEL

Format

```
SEM_APIS.DROP_SEM_MODEL(
     model_name   IN VARCHAR2,
     options       IN VARCHAR2 DEFAULT NULL,
     network_owner IN VARCHAR2 DEFAULT NULL,
     network_name  IN VARCHAR2 DEFAULT NULL);
```

> **Note:**
>
> This subprogram will be deprecated in a future release. It is recommended that you use the SEM_APIS.DROP_RDF_GRAPH subprogram instead.

Description

Drops (deletes) a semantic technology model.

Parameters

model_name
Name of the model.

options
(Reserved for future use.)

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

This procedure deletes the model from the SEM_MODEL$ view, which is described in Metadata for Models.

This procedure is the only supported way to delete a model. Do not use SQL DELETE statements with the SEM_MODEL$ view.

Only the creator of a model can delete the model.

To truncate a model instead of deleting it, use the SEM_APIS.TRUNCATE_SEM_MODEL procedure.

For information about semantic network types and options, see RDF Networks.

Examples

The following example drops the semantic technology model named `articles`.

```
EXECUTE SEM_APIS.DROP_SEM_MODEL('articles');
```

15.75 SEM_APIS.DROP_SEM_NETWORK

Format

```
SEM_APIS.DROP_SEM_NETWORK(
     cascade       IN BOOLEAN DEFAULT FALSE,
     options       IN VARCHAR2 DEFAULT NULL,
     network_owner IN VARCHAR2 DEFAULT NULL,
     network_name  IN VARCHAR2 DEFAULT NULL);
```

> **Note:**
>
> This subprogram will be deprecated in a future release. It is recommended that you use the SEM_APIS.DROP_RDF_NETWORK subprogram instead.

Description

Removes structures used for persistent storage of semantic data.

Parameters

cascade
`TRUE` drops any existing semantic technology models and rulebases, and removes structures used for persistent storage of semantic data; `FALSE` (the default) causes the operation to fail if any semantic technology models or rulebases exist.

options
(Reserved for future use.)

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

To remove structures used for persistent storage of semantic data, you must connect as a user with DBA privileges or as the owner of the schema-private network, and call this procedure.

If any version-enabled models exist, this procedure will fail regardless of the value of the `cascade` parameter.

For information about semantic network types and options, see RDF Networks.

Examples

The following example removes structures used for persistent storage of semantic data.

```
EXECUTE SEM_APIS.DROP_SEM_NETWORK;
```

15.76 SEM_APIS.DROP_SEM_SQL

Format

```
SEM_APIS.DROP_SEM_SQL;
```

Description

Drops SEM_SQL SQL Macro.

Parameters

Usage Notes

Examples

The following example drops SEM_SQL SQL Macro.

```
EXECUTE SEM_APIS.DROP_SEM_SQL;
```

15.77 SEM_APIS.DROP_SPARQL_UPDATE_TABLES

Format

```
SEM_APIS.DROP_SPARQL_UPDATE_TABLES();
```

Description

Drops the global temporary tables in the caller's schema for use with SPARQL Update operations.

Parameters

None.

Usage Notes

This procedure drops the global temporary tables that were created by the SEM_APIS.CREATE_SPARQL_UPDATE_TABLES procedure.

For more information, see Support for SPARQL Update Operations on an RDF Graph.

Examples

The following example drops the global temporary tables that had been created in the caller's schema for use with SPARQL Update operations.

```
EXECUTE SEM_APIS.DROP_SPARQL_UPDATE_TABLES;
```

15.78 SEM_APIS.DROP_SPM_TAB

Format

```
SEM_APIS.DROP_SPM_TAB (
     spm_type            IN NUMBER,
     spm_name            IN VARCHAR2,
     model_name          IN VARCHAR2,
     options             IN VARCHAR2 DEFAULT NULL,
     network_owner       IN DBMS_ID DEFAULT NULL,
     network_name        IN VARCHAR2 DEFAULT NULL);
```

> **Note:**
>
> This subprogram will be deprecated in a future release. It is recommended that you use the SEM_APIS.DROP_RESULT_TAB subprogram instead.

Description

Drops SPM table(s) defined on a given RDF model.

Parameters

spm_type
Type of the SPM table.
The value can be one of the following:

- `SEM_APIS.SPM_TYPE_SVP`
- `SEM_APIS.SPM_TYPE_MVP`
- `SEM_APIS.SPM_TYPE_PCN`
- `SEM_APIS.SPM_TYPE_ALL`
  Use of `SEM_APIS.SPM_TYPE_ALL` indicates that the target SPM tables can be of any type.

spm_name
String for use as part of the name of the SPM table. If the target is an MVP table, then specify the name of the property. Use of the value '`*`' indicates that the target is the set of all the SPM tables of the type specified by the `spm_type` parameter.

model_name
Name of the RDF model.

options
Reserved for future use.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

- This operation has a DDL semantics.

- The invoker must be the owner of the target RDF model or the RDF network or both.

Examples

The following example drops a specific SVP table named `FLHF`:

```
BEGIN
  SEM_APIS.DROP_SPM_TAB(
     spm_type      => SEM_APIS.SPM_TYPE_SVP
   , spm_name      => 'FLHF'
   , model_name    => 'M1'
   , network_owner => 'RDFUSER'
   , network_name  => 'NET1'
  );
END;
/
```

The following example drops all the currently existing SPM tables on model `M1`:

```
BEGIN
  SEM_APIS.DROP_SPM_TAB(
     spm_type      => SEM_APIS.SPM_TYPE_ALL
   , spm_name      => '*'
   , model_name    => 'M1'
   , network_owner => 'RDFUSER'
   , network_name  => 'NET1'
  );
END;
/
```

15.79 SEM_APIS.DROP_USER_INFERENCE_OBJS

Format

```
SEM_APIS.DROP_USER_INFERENCE_OBJS(
     uname         IN VARCHAR2,
     options       IN VARCHAR2 DEFAULT NULL,
     network_owner IN VARCHAR2 DEFAULT NULL,
     network_name  IN VARCHAR2 DEFAULT NULL);
```

Description

Drops (deletes) all rulebases and entailments owned by a specified database user.

Parameters

uname
Name of a database user. (This value is case-sensitive; for example, `HERMAN` and `herman` are considered different users.)

options
(Reserved for future use.)

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

You must have sufficient privileges to delete rules and rulebases for the specified user.

This procedure does not delete the database user. It deletes only RDF rulebases and entailments owned by that user.

For information about RDF network types and options, see RDF Networks.

Examples

The following example deletes all rulebases and entailments owned by user `SCOTT`.

```
EXECUTE SEM_APIS.DROP_USER_INFERENCE_OBJS('SCOTT');

PL/SQL procedure successfully completed.
```

15.80 SEM_APIS.DROP_VIRTUAL_MODEL

Format

```
SEM_APIS.DROP_VIRTUAL_MODEL(
     vm_name       IN VARCHAR2,
     options       IN VARCHAR2 DEFAULT NULL,
     network_owner IN VARCHAR2 DEFAULT NULL,
     network_name  IN VARCHAR2 DEFAULT NULL);
```

> **Note:**
>
> This subprogram will be deprecated in a future release. It is recommended that you use the SEM_APIS.DROP_RDF_GRAPH_COLLECTION subprogram instead.

Description

Drops (deletes) a virtual model.

Parameters

vm_name
Name of the virtual model to be deleted.

options
(Reserved for future use.)

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

You can use this procedure to delete a virtual model that you created using the SEM_APIS.CREATE_VIRTUAL_MODEL procedure. A virtual model is deleted automatically if any of its component models, rulebases, or entailment are deleted.

To use this procedure, you must be the owner of the specified virtual model.

For an explanation of virtual models, including usage information, see Virtual Models.

For information about semantic network types and options, see RDF Networks.

Examples

The following example deletes a virtual model named `VM1`.

```
EXECUTE sem_apis.drop_virtual_model('VM1');
```

15.81 SEM_APIS.ENABLE_CHANGE_TRACKING

Format

```
SEM_APIS.ENABLE_CHANGE_TRACKING(
     models_in     IN SEM_MODELS,
     network_owner IN VARCHAR2 DEFAULT NULL,
     network_name  IN VARCHAR2 DEFAULT NULL);
```

Description

Enables change tracking for a specified set of models.

Parameters

models_in
One or more model names. Its data type is SEM_MODELS, which has the following definition: `TABLE OF VARCHAR2(25)`

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

Change tracking must be enabled on a model before incremental inference can be enabled on any entailments that use the model.

To use this procedure, you must be the owner of the specified model or models.

If the owner of an entailment is also an owner of any underlying models, then enabling incremental inference on the entailment (by calling the SEM_APIS.ENABLE_INC_INFERENCE procedure) automatically enables change tracking on those models owned by that user.

To disable change tracking for a set of models, use the SEM_APIS.DISABLE_CHANGE_TRACKING procedure.

For an explanation of incremental inference, including usage information, see Performing Incremental Inference.

For information about RDF network types and options, see RDF Networks.

Examples

The following example enables change tracking for the `family` model.

```
EXECUTE sem_apis.enable_change_tracking(sem_models('family'));
```

15.82 SEM_APIS.ENABLE_INC_INFERENCE

Format

```
SEM_APIS.ENABLE_INC_INFERENCE(
     entailment_name IN VARCHAR2,
     network_owner   IN VARCHAR2 DEFAULT NULL,
     network_name    IN VARCHAR2 DEFAULT NULL);
```

Description

Enables incremental inference for a specified entailment (rules index).

Parameters

entailment_name
Name of the entailment for which to enable incremental inference.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

To use this procedure, you must be the owner of the specified entailment.

Before this procedure is executed, all underlying models involved in the entailment must have change tracking enabled. If the owner of the entailment is also an owner of any underlying models, calling this procedure automatically enables change tracking on those models. However, if some underlying model are not owned by the owner of the entailment, the appropriate model owners must first call the SEM_APIS.ENABLE_CHANGE_TRACKING procedure to enable change tracking on those models.

To disable incremental inference for an entailment, use the SEM_APIS.DISABLE_INC_INFERENCE procedure.

For an explanation of incremental inference, including usage information, see Performing Incremental Inference.

For information about RDF network types and options, see RDF Networks.

Examples

The following example enables incremental inference for the entailment named `RDFS_RIX_FAMILY`.

```
EXECUTE sem_apis.enable_inc_inference('rdfs_rix_family');
```

15.83 SEM_APIS.ENABLE_INMEMORY

Format

```
SEM_APIS.ENABLE_INMEMORY(
     populate_wait IN BOOLEAN,
     options       IN VARCHAR2 DEFAULT NULL,
     network_owner IN VARCHAR2 DEFAULT NULL,
     network_name  IN VARCHAR2 DEFAULT NULL);
```

Description

Loads RDF data for the RDF network into memory.

Parameters

populate_wait
Boolean value to indicate whether to wait until all RDF data is loaded into memory before finishing:

- `true`: Wait until all RDF data is loaded into memory.
- `false`: Do not wait for RDF data loading into memory.

options
Options for in-memory data population:

- The string `POPULATE_TRIPLES=F` disables populating RDF_LINK$ table data in memory. (RDF_VALUE$ table data is still populated in memory.) If this option is not specified, RDF_LINK$ table data is populated in memory by default.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

To use this procedure, you must have DBA privileges.

See the information in RDF Support for Oracle Database In-Memory.

To disable in-memory population of RDF data in the RDF network, use the SEM_APIS.DISABLE_INMEMORY.

For information about RDF network types and options, see RDF Networks.

Examples

The following example enables in-memory population of RDF data, and waits until all RDF data is loaded into memory before finishing.

```
EXECUTE SEM_APIS.ENABLE_INMEMORY(true);
```

15.84 SEM_APIS.ENABLE_INMEMORY_FOR_ENT

Format

```
SEM_APIS.ENABLE_INMEMORY_FOR_ENT(
     entailment_name IN VARCHAR2,
     network_owner   IN VARCHAR2 DEFAULT NULL,
     network_name    IN VARCHAR2 DEFAULT NULL);
```

> **Note:**
>
> This subprogram will be deprecated in a future release. It is recommended that you use the SEM_APIS.ENABLE_INMEMORY_FOR_INF_GRAPH subprogram instead.

Description

Enables in-memory population of RDF data for an entailment in a semantic network.

Parameters

entailment_name
Name of the entailment.

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

To use this procedure, you must have DBA privileges.

See the information in RDF Support for Oracle Database In-Memory.

For information about semantic network types and options, see RDF Networks.

Examples

The following example enables in-memory population of RDF data for entailment RIDX1 in the semantic network named NET1 owned by RDFUSER.

```
EXECUTE SEM_APIS.ENABLE_INMEMORY_FOR_ENT('RIDX1', network_owner=>'RDFUSER',
network_name=>'NET1');
```

15.85 SEM_APIS.ENABLE_INMEMORY_FOR_INF_GRAPH

Format

```
SEM_APIS.ENABLE_INMEMORY_FOR_INF_GRAPH(
     inferred_graph_name IN VARCHAR2,
     network_owner       IN VARCHAR2 DEFAULT NULL,
     network_name        IN VARCHAR2 DEFAULT NULL);
```

Description

Enables in-memory population of RDF data for an inferred graph in an RDF network.

Parameters

inferred_graph_name
Name of the inferred graph.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

To use this procedure, you must have DBA privileges.

See the information in RDF Support for Oracle Database In-Memory.

For information about RDF network types and options, see RDF Networks.

Examples

The following example enables in-memory population of RDF data for the inferred graph RIDX1 in the RDF network named NET1 owned by RDFUSER.

```
EXECUTE SEM_APIS.ENABLE_INMEMORY_FOR_INF_GRAPH('RIDX1', network_owner=>'RDFUSER',
network_name=>'NET1');
```

15.86 SEM_APIS.ENABLE_INMEMORY_FOR_MODEL

Format

```
SEM_APIS.ENABLE_INMEMORY_FOR_MODEL(
     model_name    IN VARCHAR2,
     network_owner IN VARCHAR2 DEFAULT NULL,
     network_name  IN VARCHAR2 DEFAULT NULL);
```

> **Note:**
>
> This subprogram will be deprecated in a future release. It is recommended that you use the SEM_APIS.ENABLE_INMEMORY_FOR_RDF_GRAPH subprogram instead.

Description

Enables in-memory population of RDF data for a model in a semantic network.

Parameters

model_name
Name of the model.

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

To use this procedure, you must have DBA privileges.

See the information in RDF Support for Oracle Database In-Memory.

For information about semantic network types and options, see RDF Networks.

Examples

The following example enables in-memory population of RDF data for model M1 in the semantic network named NET1 owned by RDFUSER.

```
EXECUTE SEM_APIS.ENABLE_INMEMORY_FOR_MODEL('M1', network_owner=>'RDFUSER',
network_name=>'NET1');
```

15.87 SEM_APIS.ENABLE_INMEMORY_FOR_RDF_GRAPH

Format

```
SEM_APIS.ENABLE_INMEMORY_FOR_RDF_GRAPH(
     rdf_graph_name IN VARCHAR2,
     network_owner  IN VARCHAR2 DEFAULT NULL,
     network_name   IN VARCHAR2 DEFAULT NULL);
```

Description

Enables in-memory population of RDF data for a graph in an RDF network.

Parameters

rdf_graph_name
Name of the RDF graph.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

To use this procedure, you must have DBA privileges.

See the information in RDF Support for Oracle Database In-Memory.

For information about RDF network types and options, see RDF Networks.

Examples

The following example enables in-memory population of RDF data for graph M1 in the RDF network named NET1 owned by RDFUSER.

```
EXECUTE SEM_APIS.ENABLE_INMEMORY_FOR_RDF_GRAPH('M1', network_owner=>'RDFUSER',
network_name=>'NET1');
```

15.88 SEM_APIS.ENABLE_NETWORK_SHARING

Format

```
SEM_APIS.ENABLE_NETWORK_SHARING(
     network_owner IN VARCHAR2,
     network_name  IN VARCHAR2,
     options       IN VARCHAR2 DEFAULT NULL);
```

Description

Enables sharing of an RDF network.

Parameters

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

options
(Reserved for future use)

Usage Notes

To use this procedure, you must have DBA privileges or be the owner of the specified network.

For information about RDF network types and options, see RDF Networks.

Examples

The following example enables sharing of the `mynetwork` schema-private network owned by database user `scott`.

```
EXECUTE SEM_APIS.ENABLE_NETWORK_SHARING('scott', 'mynetwork');
```

15.89 SEM_APIS.ESCAPE_CLOB_TERM

Format

```
SEM_APIS.ESCAPE_CLOB_TERM(
     term        IN  CLOB CHARACTER SET ANY_CS,
     utf_encode  IN NUMBER DEFAULT 1,
     options     IN VARCHAR2 DEFAULT NULL,
     max_vc_len  IN NUMBER DEFAULT 4000
     ) RETURN CLOB CHARACTER SET val%CHARSET;
```

Description

Returns the input RDF term with special characters and non-ASCII characters escaped as specified by the W3C N-Triples format (`http://www.w3.org/TR/rdf-testcases/#ntriples`).

Parameters

term
The RDF term to escape.

utf_encode
Set to 1 (the default) if non-ASCII characters and non-printable ASCII characters other than chr(8), chr(9), chr(10), chr(12), and chr(13) should be escaped. Otherwise, such characters will not be escaped.

options
Reserved for future use.

max_vc_len
The maximum allowed length of a VARCHAR RDF term - 32767 or 4000 (the default).

Usage Notes

For information about using the `DO_UNESCAPE` keyword in the `options` parameter of the SEM_MATCH table function, see Using the SEM_MATCH Table Function to Query RDF Data.

Examples

The following example escapes an input RDF term containing TAB and NEWLINE characters.

```
SELECT SEM_APIS.ESCAPE_CLOB_TERM('"abc' || chr(9) || 'def' || chr(10) || 'hij"^^<http://
www.w3.org/2001/XMLSchema#string>')
  FROM DUAL;
```

15.90 SEM_APIS.ESCAPE_CLOB_VALUE

Format

```
SEM_APIS.ESCAPE_CLOB_VALUE(
     val           IN  CLOB CHARACTER SET ANY_CS,
     start_offset  IN NUMBER   DEFAULT 1,
     end_offset    IN NUMBER   DEFAULT 0,
     utf_encode    IN NUMBER   DEFAULT 1,
     include_start IN NUMBER   DEFAULT 0,
     options       IN VARCHAR2 DEFAULT NULL,
     max_vc_len    IN NUMBER   DEFAULT 4000
     ) RETURN VARCHAR2 CHARACTER SET val%CHARSET;
```

Description

Returns the input CLOB value with special characters and non-ASCII characters escaped as specified by the W3C N-Triples format (`http://www.w3.org/TR/rdf-testcases/#ntriples`).

Parameters

val
The CLOB text to escape.

start_offset
The offset in `val` from which to start character escaping. The default (1) causes escaping to start at the first character of `val`.

end_offset
The offset in `val` from which to end character escaping. The default (0) causes escaping to continue through the end of `val`.

utf_encode
Set to 1 (the default) if non-ASCII characters and non-printable ASCII characters other than chr(8), chr(9), chr(10), chr(12), and chr(13) should be escaped. Otherwise, such characters will not be escaped.

include_start
Set to 1 if the characters in val from 1 to `start_offset` should be prefixed (prepended) to the return value. Otherwise, no such characters will be prefixed to the return value.

options
Reserved for future use.

max_vc_len
The maximum allowed length of a VARCHAR RDF term - 32767 or 4000 (the default).

Usage Notes

For information about using the `DO_UNESCAPE` keyword in the `options` parameter of the SEM_MATCH table function, see Using the SEM_MATCH Table Function to Query RDF Data.

Examples

The following example escapes an input character string containing TAB and NEWLINE characters.

```
SELECT SEM_APIS.ESCAPE_CLOB_VALUE('abc' || chr(9) || 'def' || chr(10) || 'hij')
  FROM DUAL;
```

15.91 SEM_APIS.ESCAPE_RDF_TERM

Format

```
SEM_APIS.ESCAPE_RDF_TERM(
     term        IN  VARCHAR2 CHARACTER SET ANY_CS,
     utf_encode  IN NUMBER DEFAULT 1,
     options     IN VARCHAR2 DEFAULT NULL,
     max_vc_len  IN NUMBER DEFAULT 4000
     ) RETURN VARCHAR2 CHARACTER SET val%CHARSET;
```

Description

Returns the input RDF term with special characters and non-ASCII characters escaped as specified by the W3C N-Triples format (`http://www.w3.org/TR/rdf-testcases/#ntriples`).

Parameters

term
The RDF term to escape.

utf_encode
Set to 1 (the default) if non-ASCII characters and non-printable ASCII characters other than chr(8), chr(9), chr(10), chr(12), and chr(13) should be escaped. Otherwise, such characters will not be escaped.

options
Reserved for future use.

max_vc_len
The maximum allowed length of a VARCHAR RDF term - 32767 or 4000 (the default).

Usage Notes

For information about using the `DO_UNESCAPE` keyword in the `options` parameter of the SEM_MATCH table function, see Using the SEM_MATCH Table Function to Query RDF Data.

Examples

The following example escapes an input RDF term containing TAB and NEWLINE characters.

```
SELECT SEM_APIS.ESCAPE_RDF_TERM('"abc' || chr(9) || 'def' || chr(10) || 'hij"^^<http://
www.w3.org/2001/XMLSchema#string>')
  FROM DUAL;
```

15.92 SEM_APIS.ESCAPE_RDF_VALUE

Format

```
SEM_APIS.ESCAPE_RDF_VALUE(
     val         IN VARCHAR2 CHARACTER SET ANY_CS,
     utf_encode  IN NUMBER DEFAULT 1,
     allow_long  IN NUMBER DEFAULT 0,
     options     IN VARCHAR2 DEFAULT NULL,
     max_vc_len  IN NUMBER DEFAULT 4000
     ) RETURN VARCHAR2 CHARACTER SET val%CHARSET;
```

Description

Returns the input CLOB value with special characters and non-ASCII characters escaped as specified by the W3C N-Triples format (`http://www.w3.org/TR/rdf-testcases/#ntriples`).

Parameters

val
The text to escape.

utf_encode
Set to 1 (the default) if non-ASCII characters and non-printable ASCII characters other than chr(8), chr(9), chr(10), chr(12), and chr(13) should be escaped. Otherwise, such characters will not be escaped.

allow_long
Set to 1 (default 0) if values longer than 4000 bytes should be allowed.

options
Reserved for future use.

max_vc_len
The maximum allowed length of a VARCHAR RDF term - 32767 or 4000 (the default).

Usage Notes

For information about using the `DO_UNESCAPE` keyword in the `options` parameter of the SEM_MATCH table function, see Using the SEM_MATCH Table Function to Query RDF Data.

Examples

The following example escapes an input character string containing TAB and NEWLINE characters.

```
SELECT SEM_APIS.ESCAPE_RDF_VALUE('abc' || chr(9) || 'def' || chr(10) || 'hij')
  FROM DUAL;
```

15.93 SEM_APIS.EXPORT_ENTAILMENT_STATS

Format

```
SEM_APIS.EXPORT_ENTAILMENT_STATS (
     entailment_name IN VARCHAR2,
     stattab         IN VARCHAR2,
     statid          IN VARCHAR2 DEFAULTNULL,
     cascade         IN BOOLEAN DEFAULT TRUE,
     statown         IN VARCHAR2 DEFAULT NULL,
     stat_category   IN VARCHAR2 DEFAULT 'OBJECT_STATS',
     network_owner   IN VARCHAR2 DEFAULT NULL,
     network_name    IN VARCHAR2 DEFAULT NULL);
```

Description

Exports statistics for a specified entailment and stores them in the user statistics table.

Parameters

entailment_name
Name of the entailment.

(other parameters)
See the parameter explanations for the DBMS_STATS.EXPORT_TABLE_STATS procedure in *Oracle Database PL/SQL Packages and Types Reference*, although `force` here applies to entailment statistics.
Specifying `cascade` also exports all index statistics associated with the entailment.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

See the information about the DBMS_STATS package in*Oracle Database PL/SQL Packages and Types Reference*.

See also Managing Statistics for Semantic Models and the Semantic Network.

For information about RDF network types and options, see RDF Networks.

Examples

The following example exports statistics for an entailment named `OWLTST_IDX` and stores them in a table named `STAT_TABLE`.

```
EXECUTE SEM_APIS.EXPORT_ENTAILMENT_STATS('owltst_idx', 'stat_table');
```

15.94 SEM_APIS.EXPORT_MODEL_STATS

Format

```
SEM_APIS.EXPORT_MODEL_STATS (
     model_name     IN VARCHAR2,
     stattab        IN VARCHAR2,
     statid         IN VARCHAR2 DEFAULT NULL,
     cascade        IN BOOLEAN DEFAULT TRUE,
     statown        IN VARCHAR2 DEFAULT NULL,
     stat_category IN VARCHAR2 DEFAULT 'OBJECT_STATS',
     network_owner IN VARCHAR2 DEFAULT NULL,
     network_name  IN VARCHAR2 DEFAULT NULL);
```

Description

Exports statistics for a specified model and stores them in the user statistics table.

Parameters

entailment_name
Name of the entailment.

(other parameters)
See the parameter explanations for the DBMS_STATS.EXPORT_TABLE_STATS procedure in *Oracle Database PL/SQL Packages and Types Reference*.
Specifying `cascade` also exports all index statistics associated with the model.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

See the information about the DBMS_STATS package in*Oracle Database PL/SQL Packages and Types Reference*.

See also Managing Statistics for Semantic Models and the Semantic Network.

For information about RDF network types and options, see RDF Networks.

Examples

The following example exports statistics for a model named `FAMILY` and stores them in a table named `STAT_TABLE`.

```
EXECUTE SEM_APIS.EXPORT_MODEL_STATS('family', 'stat_table');
```

15.95 SEM_APIS.EXPORT_RDFVIEW_GRAPH

Format

```
SEM_APIS.EXPORT_RDFVIEW_GRAPH(
     rdf_graph_name    IN VARCHAR2,
     rdf_table_owner   IN VARCHAR2 DEFAULT NULL,
     rdf_table_name    IN VARCHAR2 DEFAULT NULL,
```

```
     options          IN VARCHAR2 DEFAULT NULL,
     network_owner    IN VARCHAR2 DEFAULT NULL,
     network_name     IN VARCHAR2 DEFAULT NULL);
```

Description

Exports (materializes) the virtual RDF triples of an RDF view to a staging table.

Parameters

rdf_graph_name
Name of the RDF view to be exported.

rdf_table_owner
Name of the schema that owns the staging table where the RDF triples obtained from the RDF view are to be stored.

rdf_table_name
Name of the staging table where the RDF triples obtained from the RDF view are to be stored.

options
(Reserved for future use)

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

You must have the SELECT privilege for the database view SEMM_<*model_name*>.

For more information about RDF views, see RDF Views: Relational Data as RDF. For information about exporting RDF views, see Exporting Virtual Content of an RDF View into a Staging Table.

For information about RDF network types and options, see RDF Networks.

Examples

The following example exports RDF triples from RDF view `empdb_model` to the staging table SCOTT.RDFTAB.

```
BEGIN
  sem_apis.export_rdfview_graph(
    rdf_graph_name  => 'empdb_model',
    rdf_table_owner => 'SCOTT',
    rdf_table_name  => 'RDFTAB'
  );
END;
/
```

15.96 SEM_APIS.EXPORT_RDFVIEW_MODEL

Format

```
SEM_APIS.EXPORT_RDFVIEW_MODEL(
     model_name       IN VARCHAR2,
     rdf_table_owner  IN VARCHAR2 DEFAULT NULL,
```

```
     rdf_table_name    IN VARCHAR2 DEFAULT NULL,
     options           IN VARCHAR2 DEFAULT NULL,
     network_owner     IN VARCHAR2 DEFAULT NULL,
     network_name      IN VARCHAR2 DEFAULT NULL);
```

> **Note:**
>
> This subprogram will be deprecated in a future release. It is recommended that you use the SEM_APIS.EXPORT_RDFVIEW_GRAPH subprogram instead.

Description

Exports (materializes) the virtual RDF triples of an RDF view to a staging table.

Parameters

model_name
Name of the RDF view to be exported.

rdf_table_owner
Name of the schema that owns the staging table where the RDF triples obtained from the RDF view are to be stored.

rdf_table_name
Name of the staging table where the RDF triples obtained from the RDF view are to be stored.

options
(Reserved for future use)

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

You must have the SELECT privilege for the database view SEMM_<*model_name*>.

For more information about RDF views, see RDF Views: Relational Data as RDF. For information about exporting RDF views, see Exporting Virtual Content of an RDF View into a Staging Table.

For information about semantic network types and options, see RDF Networks.

Examples

The following example exports RDF triples from RDF view `empdb_model` to the staging table SCOTT.RDFTAB.

```
BEGIN
  sem_apis.export_rdfview_model(
    model_name => 'empdb_model',
    rdf_table_owner => 'SCOTT',
    rdf_table_name => 'RDFTAB'
  );
END;
/
```

15.97 SEM_APIS.GATHER_SPM_INFO

Format

```
SEM_APIS.GATHER_SPM_INFO (
    model_name          IN VARCHAR2,
    pred_info_tabname   IN DBMS_ID,
    tablespace_name     IN DBMS_ID DEFAULT NULL,
    degree              IN NUMBER DEFAULT NULL,
    options             IN VARCHAR2 DEFAULT NULL,
    network_owner       IN DBMS_ID DEFAULT NULL,
    network_name        IN VARCHAR2 DEFAULT NULL);
```

Description

Gathers information about predicate use in a given RDF model.

For more information on SPM tables content, see Creating and Managing Result Tables.

Parameters

model_name
Name of the RDF model.

pred_info_tabname
Name of the table to be created to contain the information about predicate use.

tablespace_name
Name of the target tablespace for the `pred_info_tabname` table.

degree
Degree of parallelism.

options
String specifying the options to use during the operation.
Supported option is:
`CREATE_ANYWAY=T`: Truncate the table specified in `pred_info_tabname`.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

- The `pred_info_tabname` table will be created in the invoker's schema.
- Invoker must have READ privilege for the RDF model.

Examples

The following example creates a new table `M1_PRED_INFO` in the invoker's schema. This table contains predicate use information for the specified model `M1` in the RDF network named `NET1` owned by `RDFUSER`.

```
begin
```

```
        sem_apis.gather_spm_info(
        model_name         => 'M1',
        pred_info_tabname => 'M1_PRED_INFO',
        degree             => 2,
        network_owner      => 'RDFUSER',
        network_name       => 'NET1'
        );
    end;
```

15.98 SEM_APIS.GET_CHANGE_TRACKING_INFO

Format

```
SEM_APIS.GET_CHANGE_TRACKING_INFO(
     model_name            IN VARCHAR2,
     enabled               OUT BOOLEAN,
     tracking_start_time   OUT TIMESTAMP,
     network_owner         IN VARCHAR2 DEFAULT NULL,
     network_name          IN VARCHAR2 DEFAULT NULL);
```

Description

Returns change tracking information for a model.

Parameters

model_name
Name of the semantic technology model.

enabled
Boolean value returned by the procedure: `TRUE` if change tracking is enabled for the model, or `FALSE` if change tacking is not enabled for the model.

tracking_start_time
Timestamp indicating when change tracking was enabled for the model (if it is enabled).

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

The `model_name` value must match a value in the MODEL_NAME column in the SEM_MODEL$ view, which is described in Metadata for Models.

To enable change tracking for a set of models, use the SEM_APIS.ENABLE_CHANGE_TRACKING procedure.

For an explanation of incremental inference, including usage information, see Performing Incremental Inference.

For information about RDF network types and options, see RDF Networks.

Examples

The following example displays change tracking information for a model.

```
DECLARE
  bEnabled  boolean;
  tsEnabled  timestamp;

BEGIN
  EXECUTE IMMEDIATE 'create table m1 (t SDO_RDF_TRIPLE_S)';
  sem_apis.create_sem_model('m1','m1','t');

  sem_apis.enable_change_tracking(sem_models('m1'));

  sem_apis.get_change_tracking_info('m1', bEnabled,  tsEnabled);
  dbms_output.put_line('is enabled:' || case when bEnabled then 'true' else 'false' end);
  dbms_output.put_line('enabled at:' || tsEnabled);
END;
/
```

15.99 SEM_APIS.GET_INC_INF_INFO

Format

```
SEM_APIS.GET_INC_INF_INFO(
     entailment_name     IN VARCHAR2,
     enabled             OUT BOOLEAN,
     prev_inf_start_time OUT TIMESTAMP,
     network_owner       IN VARCHAR2 DEFAULT NULL,
     network_name        IN VARCHAR2 DEFAULT NULL);
```

Description

Returns incremental inference information for an entailment.

Parameters

entailment_name
Name of the entailment.

enabled
Boolean value returned by the procedure: `TRUE` if incremental inference is enabled for the entailment, or `FALSE` if incremental inference is not enabled for the entailment.

prev_inf_start_time
Timestamp indicating when the entailment was most recently updated (if incremental inference is enabled).

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

To enable incremental inference for an entailment, use the SEM_APIS.ENABLE_INC_INFERENCE procedure.

For an explanation of incremental inference, including usage information, see Performing Incremental Inference.

For information about RDF network types and options, see RDF Networks.

Examples

The following example displays incremental inference information for an entailment.

```
DECLARE
  bEnabled boolean;
  tsEnabled timestamp;

DECLARE
  EXECUTE IMMEDIATE 'create table m1 (t SDO_RDF_TRIPLE_S)';
  sem_apis.create_sem_model('m1','m1','t');

  sem_apis.create_entailment('m1_inf',sem_models('m1'),
sem_rulebases('owlprime'),null,null,'INC=T');

  sem_apis.get_inc_inf_info('m1_inf', bEnabled,  tsEnabled);
  dbms_output.put_line('is enabled:' || case when bEnabled then 'true' else 'false'
  end);
  dbms_output.put_line('enabled at:' || tsEnabled);
END
/
```

15.100 SEM_APIS.GET_MODEL_ID

Format

```
SEM_APIS.GET_MODEL_ID(
     model_name  IN VARCHAR2
     ) RETURN NUMBER;
```

Description

Returns the model ID number of a semantic technology model.

Parameters

model_name
Name of the semantic technology model.

Usage Notes

The `model_name` value must match a value in the MODEL_NAME column in the SEM_MODEL$ view, which is described in RDF Graphs.

Examples

The following example returns the model ID number for the model named `articles`. (This example is an excerpt from Example 1-129 in Example: Journal Article Information.)

```
SELECT SEM_APIS.GET_MODEL_ID('articles') AS model_id FROM DUAL;

  MODEL_ID
----------
         1
```

15.101 SEM_APIS.GET_MODEL_NAME

Format

```
SEM_APIS.GET_MODEL_NAME(
     model_id  IN NUMBER
     ) RETURN VARCHAR2;
```

Description

Returns the model name of a semantic technology model.

Parameters

model_id
ID number of the semantic technology model.

Usage Notes

The `model_id` value must match a value in the MODEL_ID column in the SEM_MODEL$ view, which is described in RDF Graphs.

Examples

The following example returns the model ID number for the model with the ID value of 1. This example is an excerpt from Example 1-129 in Example: Journal Article Information.)

```
SQL> SELECT SEM_APIS.GET_MODEL_NAME(1) AS model_name FROM DUAL;

MODEL_NAME
--------------------------------------------------------------------------------
ARTICLES
```

15.102 SEM_APIS.GET_PLAN_COST

Format

```
SEM_APIS.GET_PLAN_COST(
    query  IN CLOB
    ) RETURN NUMBER;
```

Description

Gets the cost of the execution plan for the query.

Parameters

query
The input query string.

Usage Notes

Examples

The following example gets the execution plan cost of the query.

```
SQL> SELECT sem_apis.get_plan_cost(q'[SELECT x, y
  2  FROM TABLE(sem_match(
  3   '{?x <email> ?y}',
  4   sem_models('m1'), null, null, null, null
  5  ,' ',null,null,'RDFUSER','MYNET')) order by 1,2]') pcost FROM DUAL;
               PCOST
--------------------
                   3
1 row selected.
```

15.103 SEM_APIS.GET_SQL

Format

```
SEM_APIS.GET_SQL(
    sparql_query  IN CLOB,
    models        IN RDF_MODELS DEFAULT NULL,
    rulebases     IN RDF_RULEBASES DEFAULT NULL,
    aliases       IN RDF_ALIASES DEFAULT NULL,
    index_status  IN VARCHAR2 DEFAULT NULL,
    options       IN VARCHAR2 DEFAULT NULL
    graphs        IN RDF_GRAPHS DEFAULT NULL,
    named_graphs  IN RDF_GRAPHS DEFAULT NULL,
    network_owner IN VARCHAR2 DEFAULT NULL,
    network_name  IN VARCHAR2 DEFAULT NULL) RETURN CLOB;
```

Description

Translates a SPARQL query into a SQL query string that can be executed by an application program.

Parameters

sparql_query
A string literal with one or more triple patterns, usually containing variables.

models
The model or models to use.

rulebases
One or more rulebases whose rules are to be applied to the query.

aliases
One or more namespaces to be used for expansion of qualified names in the query pattern.

index_status
The status of the relevant entailment for this query.

options
Options that can affect the results of queries.

graphs
The set of named graphs from which to construct the default graph for the query.

named_graphs
The set of named graphs that can be matched by a GRAPH clause.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

Before using this procedure, ensure you understand the material in Using the SEM_APIS.GET_SQL Function and SEM_SQL SQL Macro to Query RDF Data.

For information about semantic network types and options, see RDF Networks.

Examples

The following example translates a SPARQL query into a SQL query string.

```
EXECUTE SEM_APIS.GET_SQL('SELECT ?s  ?o { ?s <http://www.w3.org/1999/02/22-
rdf-syntax-ns#type> ?o }',
sem_models('m1'),null,null,null,'
',null,null,network_owner=>'RDFUSER',network_name=>'MYNET');
```

15.104 SEM_APIS.GET_TRIPLE_ID

Format

```
SEM_APIS.GET_TRIPLE_ID(
     model_id  IN NUMBER,
     subject   IN VARCHAR2,
     property  IN VARCHAR2,
     object    IN VARCHAR2
     ) RETURN VARCHAR2;
```

or

```
SEM_APIS.GET_TRIPLE_ID(
     model_name IN VARCHAR2,
     subject    IN VARCHAR2,
     property   IN VARCHAR2,
     object     IN VARCHAR2
     ) RETURN VARCHAR2;
```

Description

Returns the ID of a triple in the specified semantic technology model, or a null value if the triple does not exist.

Parameters

model_id
ID number of the semantic technology model. Must match a value in the MODEL_ID column of the SEM_MODEL$ view, which is described in RDF Graphs.

model_name
Name of the semantic technology model. Must match a value in the MODEL_NAME column of the SEM_MODEL$ view, which is described in RDF Graphs.

subject
Subject. Must match a value in the VALUE_NAME column of the RDF_VALUE$ table, which is described in Statements.

property
Property. Must match a value in the VALUE_NAME column of the RDF_VALUE$ table, which is described in Statements.

object
Object. Must match a value in the VALUE_NAME column of the RDF_VALUE$ table, which is described in Statements.

Usage Notes

This function has two formats, enabling you to specify the semantic technology model by its model number or its name.

Examples

The following example returns the ID number of a triple. (This example is an excerpt from Example 1-129 in Example: Journal Article Information.)

```
SELECT SEM_APIS.GET_TRIPLE_ID(
  'articles',
  'http://nature.example.com/Article2',
  'http://purl.org/dc/terms/references',
  'http://nature.example.com/Article3') AS RDF_triple_id FROM DUAL;

RDF_TRIPLE_ID
--------------------------------------------------------------------------------
2_9F2BFF05DA0672E_90D25A8B08C653A_46854582F25E8AC5
```

15.105 SEM_APIS.GETV$DATETIMETZVAL

Format

```
SEM_APIS.GETV$DATETIMETZVAL(
     value_type     IN VARCHAR2,
     vname_prefix   IN VARCHAR2,
     vname_suffix   IN VARCHAR2,
     literal_type   IN VARCHAR2,
     language_type  IN VARCHAR2,
     ) RETURN NUMBER;
```

Description

Returns a TIMESTAMP WITH TIME ZONE value for xsd:dateTime typed literals, and returns a null value for all other RDF terms. Greenwich Mean Time is used as the default time zone for xsd:dateTime values without time zones.

Parameters

value_type
Type of the RDF term.

vname_prefix
Prefix value of the RDF term.

vname_suffix
Suffix value of the RDF term.

literal_type
Literal type of the RDF term.

language_type
Language type of the RDF term.

Usage Notes

For better performance, consider creating a function-based index on this function. For more information, see Function-Based Indexes for FILTER Constructs Involving Typed Literals.

Examples

The following example returns TIMESTAMP WITH TIME ZONE values for all xsd:dateTime literals in the RDF_VALUE$ table:

```
SELECT SEM_APIS.GETV$DATETIMETZVAL(value_type, vname_prefix, vname_suffix,
  literal_type, language_type)
  FROM RDF_VALUE$;
```

15.106 SEM_APIS.GETV$DATETZVAL

Format

```
SEM_APIS.GETV$DATETZVAL(
     value_type     IN VARCHAR2,
     vname_prefix   IN VARCHAR2,
     vname_suffix   IN VARCHAR2,
     literal_type   IN VARCHAR2,
     language_type  IN VARCHAR2,
     ) RETURN TIMESTAMP WITH TIME ZONE;
```

Description

Returns a TIMESTAMP WITH TIME ZONE value for xsd:date typed literals, and returns a null value for all other RDF terms. Greenwich Mean Time is used as the default time zone for xsd:date values without time zones.

Parameters

value_type
Type of the RDF term.

vname_prefix
Prefix value of the RDF term.

vname_suffix
Suffix value of the RDF term.

literal_type
Literal type of the RDF term.

language_type
Language type of the RDF term.

Usage Notes

For better performance, consider creating a function-based index on this function. For more information, see Function-Based Indexes for FILTER Constructs Involving Typed Literals.

Examples

The following example returns TIMESTAMP WITH TIME ZONE values for all xsd:date literals in the RDF_VALUE$ table:

```
SELECT SEM_APIS.GETV$DATETZVAL(value_type, vname_prefix, vname_suffix,
  literal_type, language_type)
  FROM RDF_VALUE$;
```

15.107 SEM_APIS.GETV$GEOMETRYVAL

Format

```
SEM_APIS.GETV$GEOMETRYVAL(
     value_type    IN VARCHAR2,
     vname_prefix  IN VARCHAR2,
     vname_suffix  IN VARCHAR2,
     literal_type  IN VARCHAR2,
     language_type IN VARCHAR2,
     long_value    IN CLOB,
     srid          IN NUMBER,
     ) RETURN SDO_GEOMETRY;
```

Description

Returns an SDO_GEOMETRY object in the spatial reference system identified by an input SRID for ogc:wktLiteral or ogc:gmlLiteral typed literals, and returns a null value for all other RDF terms.

Parameters

value_type
Type of the RDF term.

vname_prefix
Prefix value of the RDF term.

vname_suffix
Suffix value of the RDF term.

literal_type
Literal type of the RDF term.

language_type
Language type of the RDF term.

long_value
CLOB value for long literals.

srid
Target coordinate system (spatial reference system) identifier for the SDO_GEOMETRY object to be returned.

Usage Notes

ogc:wktLiteral and ogc:gmlLiteral values encode spatial reference system information in the literal value itself (referred to as the **source SRID**).

If the `srid` parameter value (the target SRID) is different from the source SRID, the newly created SDO_GEOMETRY object is transformed to the target SRID before it is returned.

This operation can be expensive in terms of performance.

For information about the SDO_GEOMETRY type (including SRID values), see *Oracle Spatial Developer's Guide*.

Examples

The following example returns SDO_GEOMETRY values in the WGS84 (Longitude, Latitude) spatial reference system (SRID 8307) for all geometry literals in the RDF_VALUE$ table:

```
SELECT SEM_APIS.GETV$GEOMETRYVAL(value_type, vname_prefix, vname_suffix,
  literal_type, language_type, long_value, 8307)
  FROM RDF_VALUE$;
```

15.108 SEM_APIS.GETV$NUMERICVAL

Format

```
SEM_APIS.GETV$NUMERICVAL(
     value_type     IN VARCHAR2,
     vname_prefix   IN VARCHAR2,
     vname_suffix   IN VARCHAR2,
     literal_type   IN VARCHAR2,
     language_type  IN VARCHAR2,
     ) RETURN NUMBER;
```

Description

Returns a numeric value for XML Schema numeric typed literals, and returns a null value for all other RDF terms.

Parameters

value_type
Type of the RDF term.

vname_prefix
Prefix value of the RDF term.

vname_suffix
Suffix value of the RDF term.

literal_type
Literal type of the RDF term.

language_type
Language type of the RDF term.

Usage Notes

For better performance, consider creating a function-based index on this function. For more information, see Function-Based Indexes for FILTER Constructs Involving Typed Literals.

Examples

The following example returns numeric values for all numeric literals in the RDF_VALUE$ table:

```
SELECT SEM_APIS.GETV$NUMERICVAL(value_type, vname_prefix, vname_suffix,
  literal_type, language_type)
  FROM RDF_VALUE$;
```

15.109 SEM_APIS.GETV$STRINGVAL

Format

```
SEM_APIS.GETV$STRINGVAL(
     value_type     IN VARCHAR2,
     vname_prefix   IN VARCHAR2,
     vname_suffix   IN VARCHAR2,
     literal_type   IN VARCHAR2,
     language_type  IN VARCHAR2,
     ) RETURN TIMESTAMP WITH TIME ZONE;
```

Description

Returns a VARCHAR2 string of the lexical form of plain literals and xsd:string typed literals, and returns a null value for all other RDF terms. CHR(0) is returned for empty literals.

Parameters

value_type
Type of the RDF term.

vname_prefix
Prefix value of the RDF term.

vname_suffix
Suffix value of the RDF term.

literal_type
Literal type of the RDF term.

language_type
Language type of the RDF term.

Usage Notes

For better performance, consider creating a function-based index on this function. For more information, see Function-Based Indexes for FILTER Constructs Involving Typed Literals.

Examples

The following example returns lexical values for all plain literals and xsd:string literals in the RDF_VALUE$ table:

```
SELECT SEM_APIS.GETV$STRINGVAL(value_type, vname_prefix, vname_suffix,
  literal_type, language_type)
  FROM RDF_VALUE$;
```

15.110 SEM_APIS.GETV$TIMETZVAL

Format

```
SEM_APIS.GETV$TIMETZVAL(
     value_type     IN VARCHAR2,
     vname_prefix   IN VARCHAR2,
     vname_suffix   IN VARCHAR2,
     literal_type   IN VARCHAR2,
     language_type  IN VARCHAR2,
     ) RETURN TIMESTAMP WITH TIME ZONE;
```

Description

Returns a TIMESTAMP WITH TIME ZONE value for xsd:time typed literals, and returns a null value for all other RDF terms. Greenwich Mean Time is used as the default time zone for xsd:time values without time zones. 2009-06-26 is used as the default date in all the generated TIMESTAMP WITH TIME ZONE values.

Parameters

value_type
Type of the RDF term.

vname_prefix
Prefix value of the RDF term.

vname_suffix
Suffix value of the RDF term.

literal_type
Literal type of the RDF term.

language_type
Language type of the RDF term.

Usage Notes

For better performance, consider creating a function-based index on this function. For more information, see Function-Based Indexes for FILTER Constructs Involving Typed Literals.

Because xsd:time values include only a time but not a date, the returned TIMESTAMP WITH TIME ZONE values (which include a date component) have 2009-06-26 added as the date. This is done so that the returned values can be indexed internally, and so that the date is the same for all of them.

Examples

The following example returns TIMESTAMP WITH TIME ZONE values (using the default 2009-06-26 for the date) for all xsd:time literals in the RDF_VALUE$ table. (

```
SELECT SEM_APIS.GETV$DATETIMETZVAL(value_type, vname_prefix, vname_suffix,
  literal_type, language_type)
  FROM RDF_VALUE$;
```

15.111 SEM_APIS.GRANT_MODEL_ACCESS_PRIV

Format

```
SEM_APIS.GRANT_MODEL_ACCESS_PRIV(
     model_name      IN VARCHAR2,
     user_name       IN VARCHAR2,
     privilege       IN VARCHAR2,
     user_view       IN VARCHAR2 DEFAULT NULL,
     options         IN VARCHAR2 DEFAULT NULL,
     network_owner   IN VARCHAR2 DEFAULT NULL,
     network_name    IN VARCHAR2 DEFAULT NULL);
```

> **Note:**
>
> This subprogram will be deprecated in a future release. It is recommended that you use the SEM_APIS.GRANT_RDF_GRAPH_ACCESS_PRIV subprogram instead.

Description

Grants access privilege on a model or entailment.

Parameters

model_name
Name of the model.

user_name
Database user that is recipient of this privilege.

privilege
Specifies the type of privilege that is granted. Currently allowed values include the following:

- `QUERY`: Query the model using SPARQL

- SELECT, READ: Retrieve model content using SQL. The source for the content is the RDFT_<model> view in the network owner's schema or the view name, if any, specified for the user_view parameter.
- INSERT, UPDATE, DELETE: Perform SPARQL Update (DML) operations on the model or SQL DML operations. For SQL DML, the target object is the RDFT_<model> view in the network owner's schema.

> **Note:**
>
> QUERY is the only valid choice if the model is not a regular model (that is, not created using sem_apis.create_sem_model).

user_view
Applicable to schema-private network only. If a view was created on the RDFT_<model> view at model creation time using sem_apis.create_sem_model or later, privilege is granted on that view.

options
If user specifies the word ENTAILMENT as part of the string value, then the specified model_name is taken as the name of an entailment (rules index). (Additional words or phrases may be allowed in future.)

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

The recipient must already have query-only or full access to the semantic network (which guarantees access to dictionary tables, but not individual models). This operation grants access to the specified model.

Examples

The following example grants privilege to database user USER1 to use SPARQL query against a semantic technology model named articles in the schema-private network NET1 owned by database user RDFUSER. (This example refers to the model described in Example 1-129.)

```
EXECUTE SEM_APIS.GRANT_MODEL_ACCESS_PRIV('articles', 'USER1', 'QUERY',
network_owner=>'RDFUSER', network_name=>'NET1');
```

15.112 SEM_APIS.GRANT_MODEL_ACCESS_PRIVS

Format

```
SEM_APIS.GRANT_MODEL_ACCESS_PRIVS(
     model_name      IN VARCHAR2,
     user_name       IN VARCHAR2,
     priv_list       IN SYS.ODCIVARCHAR2LIST,
     user_view       IN VARCHAR2 DEFAULT NULL,
     options         IN VARCHAR2 DEFAULT NULL,
     network_owner   IN VARCHAR2 DEFAULT NULL,
     network_name    IN VARCHAR2 DEFAULT NULL);
```

> **Note:**
>
> This subprogram will be deprecated in a future release. It is recommended that you use the SEM_APIS.GRANT_RDF_GRAPH_ACCESS_PRIVS subprogram instead.

Description

Grants access privileges on a model or entailment.

Parameters

model_name
Name of the model.

user_name
Database user that is recipient of this privilege.

priv_list
Specifies the list of privileges that are granted. Currently allowed values include the following:

- `QUERY`: Query the model using SPARQL
- `SELECT`, `READ`: Retrieve model content using SQL. The source for the content is the `RDFT_<model>` view in the network owner's schema or the view name, if any, specified for the `user_view` parameter.
- `INSERT`, `UPDATE`, `DELETE`: Perform SPARQL Update (DML) operations on the model or SQL DML operations. For SQL DML, the target object is the `RDFT_<model>` view in the network owner's schema.

> **Note:**
>
> `QUERY` is the only valid choice if the model is not a regular model (that is, not created using `sem_apis.create_sem_model`).

user_view
Applicable to schema-private network only. If a view was created on the `RDFT_<model>` view at model creation time using `sem_apis.create_sem_model` or later, privileges are granted on that view.

options
If user specifies the word `ENTAILMENT` as part of the string value, then the specified `model_name` is taken as the name of an entailment (rules index).(Additional words or phrases may be allowed in future.)

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

The recipient must already have query-only or full access to the semantic network (which guarantees access to dictionary tables, but not individual models). This operation grants access to the specified model.

Examples

The following example grants privileges to perform DML operations against a semantic technology model named `articles` in the schema-private network `NET1` owned by database user `RDFUSER`. (This example refers to the model described in Example 1-129.)

```
EXECUTE SEM_APIS.GRANT_MODEL_ACCESS_PRIVS('articles', 'USER1',
sys.odcivarchar2list('INSERT','UPDATE','DELETE'), network_owner=>'RDFUSER',
network_name=>'NET1');
```

15.113 SEM_APIS.GRANT_NETWORK_ACCESS_PRIVS

Format

```
SEM_APIS.GRANT_NETWORK_ACCESS_PRIVS(
     network_owner   IN VARCHAR2,
     network_name    IN VARCHAR2,
     network_user    IN VARCHAR2,
     options         IN VARCHAR2 default NULL);
```

Description

Grants query-only or full access privileges to a database user other than the owner of a schema-private RDF network.

Parameters

network_owner
Owner of the RDF network. (Cannot be MDSYS.)

network_name
Name of the RDF network. (Must be a schema-private network.)

network_user
Database user (other than the network owner) to which to grant access privileges to the network.

options
String specifying options for access using the form *OPTION_NAME=option_value*. By default, full access privileges are given; but to give query-only access, specify `QUERY_ONLY=T` for the option value.

Usage Notes

You must have DBA privileges or be the owner of the specified network to call this procedure.

For information about RDF network types and options, see RDF Networks.

Examples

The following example grants full access on the `mynet1` network owned by `scott` to `rdfuser1`.

```
EXECUTE SEM_APIS.GRANT_NETWORK_ACCESS_PRIVS('scott','mynet1','rdfuser1');
```

The following example grants query-only access on the `mynet1` network owned by `scott` to `rdfuser2`.

```
EXECUTE SEM_APIS.GRANT_NETWORK_ACCESS_PRIVS('scott','mynet1','rdfuser2', options=>'
QUERY_ONLY=T ');
```

15.114 SEM_APIS.GRANT_NETWORK_SHARING_PRIVS

Format

```
SEM_APIS.GRANT_NETWORK_SHARING_PRIVS(
     network_owner  IN VARCHAR2,
     options        IN VARCHAR2 default NULL);
```

> **Note:**
>
> You can skip the `GRANT_NETWORK_SHARING_PRIVS` procedure if your Oracle Database version is 23.4.

Description

Grants to a database user the privileges required for sharing, with other database users, any schema-private networks owned (currently or in the future) by the database user.

Parameters

network_owner
Owner of the RDF network. (See Table 1-2.)

options
(Reserved for future use)

Usage Notes

You must have DBA privileges to call this procedure.

For information about RDF network types and options, see RDF Networks.

Examples

The following example grants to database user `scott` the privileges for sharing any schema-private networks that this user owns or will own.

```
EXECUTE SEM_APIS.GRANT_NETWORK_SHARING_PRIVS('scott');
```

15.115 SEM_APIS.GRANT_RDF_GRAPH_ACCESS_PRIV

Format

```
SEM_APIS.GRANT_RDF_GRAPH_ACCESS_PRIV(
     rdf_graph_name  IN VARCHAR2,
     user_name       IN VARCHAR2,
     privilege       IN VARCHAR2,
     user_view       IN VARCHAR2 DEFAULT NULL,
```

```
    options         IN VARCHAR2 DEFAULT NULL,
    network_owner   IN VARCHAR2 DEFAULT NULL,
    network_name    IN VARCHAR2 DEFAULT NULL);
```

Description

Grants access privilege on an RDF graph or inferred graph.

Parameters

rdf_graph_name
Name of the RDF graph.

user_name
Database user that is recipient of this privilege.

privilege
Specifies the type of privilege that is granted. Currently allowed values include the following:

- `QUERY`: Query the RDF graph using SPARQL
- `SELECT`, `READ`: Retrieve the RDF graph content using SQL. The source for the content is the `RDFT_<rdf_graph>` view in the network owner's schema or the view name, if any, specified for the `user_view` parameter.
- `INSERT`, `UPDATE`, `DELETE`: Perform SPARQL Update (DML) operations on the RDF graph or SQL DML operations. For SQL DML, the target object is the `RDFT_<rdf_graph_name>` view in the network owner's schema.

> **Note:**
>
> `QUERY` is the only valid choice if the RDF graph is not a regular RDF graph (that is, not created using `sem_apis.create_rdf_graph`).

user_view
Applicable to schema-private network only. If a view was created on the `RDFT_<rdf_graph_name>` view at RDF graph creation time using `sem_apis.create_rdf_graph` or later, privilege is granted on that view.

options
If user specifies the word `ENTAILMENT` as part of the string value, then the specified `rdf_graph_name` is taken as the name of an inferred graph (rules index). (Additional words or phrases may be allowed in future.)

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

The recipient must already have query-only or full access to the RDF network (which guarantees access to dictionary tables, but not individual RDF graphs). This operation grants access to the specified RDF graph.

Examples

The following example grants privilege to database user `USER1` to use SPARQL query against a semantic technology RDF graph named `articles` in the schema-private network `NET1` owned by database user `RDFUSER`. (This example refers to the RDF graph described in Example 1-129.)

```
EXECUTE SEM_APIS.GRANT_RDF_GRAPH_ACCESS_PRIV('articles', 'USER1', 'QUERY',
network_owner=>'RDFUSER', network_name=>'NET1');
```

15.116 SEM_APIS.GRANT_RDF_GRAPH_ACCESS_PRIVS

Format

```
SEM_APIS.GRANT_RDF_GRAPH_ACCESS_PRIVS(
     rdf_graph_name   IN VARCHAR2,
     user_name        IN VARCHAR2,
     priv_list        IN SYS.ODCIVARCHAR2LIST,
     user_view        IN VARCHAR2 DEFAULT NULL,
     options          IN VARCHAR2 DEFAULT NULL,
     network_owner    IN VARCHAR2 DEFAULT NULL,
     network_name     IN VARCHAR2 DEFAULT NULL);
```

Description

Grants access privileges on an RDF graph or inferred graph.

Parameters

rdf_graph_name
Name of the RDF graph.

user_name
Database user that is recipient of this privilege.

priv_list
Specifies the list of privileges that are granted. Currently allowed values include the following:

- `QUERY`: Query the RDF graph using SPARQL
- `SELECT`, `READ`: Retrieve the RDF graph content using SQL. The source for the content is the `RDFT_<rdf_graph_name>` view in the network owner's schema or the view name, if any, specified for the `user_view` parameter.
- `INSERT`, `UPDATE`, `DELETE`: Perform SPARQL Update (DML) operations on the RDF graph or SQL DML operations. For SQL DML, the target object is the `RDFT_<rdf_graph_name>` view in the network owner's schema.

> **Note:**
>
> `QUERY` is the only valid choice if the RDF graph is not a regular graph (that is, not created using `sem_apis.create_rdf_graph`).

user_view
Applicable to schema-private network only. If a view was created on the `RDFT_<rdf_graph_name>` view at RDF graph creation time using `sem_apis.create_rdf_graph` or later, privileges are granted on that view.

options
If user specifies the word `ENTAILMENT` as part of the string value, then the specified `rdf_graph_name` is taken as the name of an inferred graph (rules index).(Additional words or phrases may be allowed in future.)

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

The recipient must already have query-only or full access to the RDF network (which guarantees access to dictionary tables, but not individual RDF graphs). This operation grants access to the specified RDF graph.

Examples

The following example grants privileges to perform DML operations against a semantic technology RDF graph named `articles` in the schema-private network `NET1` owned by database user `RDFUSER`. (This example refers to the RDF graph described in Example 1-129.)

```
EXECUTE SEM_APIS.GRANT_RDF_GRAPH_ACCESS_PRIVS('articles', 'USER1',
sys.odcivarchar2list('INSERT','UPDATE','DELETE'), network_owner=>'RDFUSER',
network_name=>'NET1');
```

15.117 SEM_APIS.IMPORT_ENTAILMENT_STATS

Format

```
SEM_APIS.IMPORT_ENTAILMENT_STATS (
     entailment_name  IN VARCHAR2,
     stattab          IN VARCHAR2,
     statid           IN VARCHAR2 DEFAULT NULL,
     cascade          IN BOOLEAN DEFAULT TRUE,
     statown          IN VARCHAR2 DEFAULT NULL,
     no_invalidate    IN BOOLEAN DEFAULT FALSE,
     force            IN BOOLEAN DEFAULT FALSE,
     stat_category    IN VARCHAR2 DEFAULT 'OBJECT_STATS',
     network_owner    IN VARCHAR2 DEFAULT NULL,
     network_name     IN VARCHAR2 DEFAULT NULL);
```

Description

Retrieves statistics for an entailment from a user statistics table and stores them in the dictionary.

Parameters

entailment_name
Name of the entailment.

(other parameters)
See the parameter explanations for the DBMS_STATS.IMPORT_TABLE_STATS procedure in *Oracle Database PL/SQL Packages and Types Reference*, although `force` here applies to entailment statistics.
Specifying `cascade` also exports all index statistics associated with the model.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

See the information about the DBMS_STATS package in*Oracle Database PL/SQL Packages and Types Reference*.

See also Managing Statistics for the RDF Graphs and RDF Network.

For information about RDF network types and options, see RDF Networks.

Examples

The following example imports statistics for an entailment named `OWLTST_IDX` from a table named `STAT_TABLE`.

```
EXECUTE SEM_APIS.IMPORT_ENTAILMENT_STATS('owltst_idx', 'stat_table');
```

15.118 SEM_APIS.IMPORT_MODEL_STATS

Format

```
SEM_APIS.IMPORT_MODEL_STATS (
     model_name     IN VARCHAR2,
     stattab        IN VARCHAR2,
     statid         IN VARCHAR2 DEFAULT NULL,
     cascade        IN BOOLEAN DEFAULT TRUE,
     statown        IN VARCHAR2 DEFAULT NULL,
     no_invalidate  IN BOOLEAN DEFAULT FALSE,
     force          IN BOOLEAN DEFAULT FALSE,
     stat_category  IN VARCHAR2 DEFAULT 'OBJECT_STATS,
     network_owner  IN VARCHAR2 DEFAULT NULL,
     network_name   IN VARCHAR2 DEFAULT NULL);
```

Description

Retrieves statistics for a specified model from a user statistics table and stores them in the dictionary.

Parameters

model_name
Name of the entailment.

(other parameters)
See the parameter explanations for the DBMS_STATS.IMPORT_TABLE_STATS procedure in *Oracle Database PL/SQL Packages and Types Reference*.
Specifying `cascade` also imports all index statistics associated with the model.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

See the information about the DBMS_STATS package in*Oracle Database PL/SQL Packages and Types Reference*.

See also Managing Statistics for Semantic Models and the Semantic Network.

For information about RDF network types and options, see RDF Networks.

Examples

The following example imports statistics for a model named `FAMILY` from a table named `STAT_TABLE`, and stores them in the dictionary.

```
EXECUTE SEM_APIS.IMOPRT_MODEL_STATS('family', 'stat_table');
```

15.119 SEM_APIS.IS_TRIPLE

Format

```
SEM_APIS.IS_TRIPLE(
     model_id  IN NUMBER,
     subject   IN VARCHAR2,
     property  IN VARCHAR2,
     object    IN VARCHAR2) RETURN VARCHAR2;
```

or

```
SEM_APIS.IS_TRIPLE(
     model_name IN VARCHAR2,
     subject    IN VARCHAR2,
     property   IN VARCHAR2,
     object     IN VARCHAR2) RETURN VARCHAR2;
```

Description

Checks if a statement is an existing triple in the specified model in the database.

Parameters

model_id
ID number of the semantic technology model. Must match a value in the MODEL_ID column of the SEM_MODEL$ view, which is described in Metadata for Models.

model_name
Name of the semantic technology model. Must match a value in the MODEL_NAME column of the SEM_MODEL$ view, which is described in Metadata for Models.

subject
Subject. Must match a value in the VALUE_NAME column of the RDF_VALUE$ table, which is described in Statements.

property
Property. Must match a value in the VALUE_NAME column of the RDF_VALUE$ table, which is described in Statements.

object
Object. Must match a value in the VALUE_NAME column of the RDF_VALUE$ table, which is described in Statements.

Usage Notes

This function returns the string value `FALSE`, `TRUE`, or `TRUE (EXACT)`:

- `FALSE` means that the statement is not a triple in the specified model the database.
- `TRUE` means that the statement matches the value of a triple or is the canonical representation of the value of a triple in the specified model the database.
- `TRUE (EXACT)` means that the specified `subject`, `property`, and `object` values have exact matches in a triple in the specified model in the database.

Examples

The following example checks if a statement is a triple in the database. In this case, there is an exact match. (This example is an excerpt from Example 1-129 in Example: Journal Article Information.)

```
SELECT SEM_APIS.IS_TRIPLE(
  'articles',
  'http://nature.example.com/Article2',
  'http://purl.org/dc/terms/references',
  'http://nature.example.com/Article3') AS is_triple FROM DUAL;

IS_TRIPLE
--------------------------------------------------------------------------------
TRUE (EXACT)
```

15.120 SEM_APIS.LOAD_INTO_STAGING_TABLE

Format

```
SEM_APIS.LOAD_INTO_STAGING_TABLE(
     stagong_table      IN VARCHAR2,
     source_table       IN VARCHAR2,
     input_format       IN VARCHAR2 DEFAULT NULL,
     parallel           IN INTEGER DEFAULT NULL,
     staging_table_owner IN VARCHAR2 DEFAULT NULL,
     source_table_owner  IN VARCHAR DEFAULT NULL,
     flags              IN VARCHAR DEFAULT NULL);
```

Description

Loads data into a staging table from an external table mapped to an N-Triple or N-Quad format input file.

Parameters

staging_table
Name of the staging table.

source_table
Name of the source external table.

input_format
Format of the input file mapped by the source external table: `N-TRIPLE` or `N-QUAD`

parallel
Degree of parallelism to use during the load.

staging_table_owner
Owner for the staging table being created. If not specified, the invoker is assumed to be the owner.

source_table_owner
Owner for the source table. If not specified, the invoker is assumed to be the owner.

flags
(Reserved for future use)

Usage Notes

For more information and an example, see Loading N-Quad Format Data into a Staging Table Using an External Table.

Examples

The following example loads the staging table. (This example is an excerpt from Example 1-109 in Loading N-Quad Format Data into a Staging Table Using an External Table.)

```
BEGIN
  sem_apis.load_into_staging_table(
    staging_table => 'STAGE_TABLE'
   ,source_table  => 'stage_table_source'
   ,input_format  => 'N-QUAD');
END;
```

15.121 SEM_APIS.LOOKUP_ENTAILMENT

Format

```
SEM_APIS.LOOKUP_ENTAILMENT (
     models      IN SEM_MODELS,
     rulebases   IN SEM_RULEBASES
     ) RETURN VARCHAR2;
```

Description

Returns the name of the entailment (rules index) based on the specified models and rulebases.

Parameters

models
One or more model names. Its data type is SEM_MODELS, which has the following definition: `TABLE OF VARCHAR2(25)`

rulebases
One or more rulebase names. Its data type is SEM_RULEBASES, which has the following definition: `TABLE OF VARCHAR2(25)` Rules and rulebases are explained in Inferencing: Rules and Rulebases.

Usage Notes

For a rulebase index to be returned, it must be based on all specified models and rulebases.

Examples

The following example finds the entailment that is based on the `family` model and the `RDFS` and `family_rb` rulebases. (It is an excerpt from Example 1-130 in Example: Family Information.)

```
SELECT SEM_APIS.LOOKUP_ENTAILMENT(SEM_MODELS('family'),
  SEM_RULEBASES('RDFS','family_rb')) AS lookup_entailment FROM DUAL;

LOOKUP_ENTAILMENT
--------------------------------------------------------------------------------
RDFS_RIX_FAMILY
```

15.122 SEM_APIS.MERGE_MODELS

Format

```
SEM_APIS.MERGE_MODELS(
     source_model          IN VARCHAR2,
     destination_model     IN VARCHAR2,
     rebuild_apptab_index IN BOOLEAN DEFAULT TRUE,
     drop_source_model     IN BOOLEAN DEFAULT FALSE,
     options               IN VARCHAR2 DEFAULT NULL,
     network_owner         IN VARCHAR2 DEFAULT NULL,
     network_name          IN VARCHAR2 DEFAULT NULL);
```

> **Note:**
>
> This subprogram will be deprecated in a future release. It is recommended that you use the SEM_APIS.MERGE_RDF_GRAPHS subprogram instead.

Description

Inserts the content from a source model into a destination model, and updates the destination application table.

Parameters

source_model
Name of the source model.

destination_model
Name of the destination model.

rebuild_apptab_index
`TRUE` causes indexes on the destination application table to be rebuilt after the models are merged; `FALSE` does not rebuild any indexes.

drop_source_model
`TRUE` causes the source model (`source_model`) to be deleted after the models are merged; `FALSE` (the default) does not delete the source model.

options
A comma-delimited string of options that overrides the default behavior of the procedure. Currently, only the `DOP` (degree of parallelism) option is supported, to enable parallel execution of this procedure and to specify the degree of parallelism to be associated with the operation.

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

Before you merge any models, if you are using positional parameters, check to be sure that you are specifying the correct models for the first and second parameters (source model for the first, destination model for the second). This is especially important if you plan to specify `drop_source_model=TRUE`.

If appropriate, make copies of the destination model or both models before performing the merge. To make a copy of a model, use SEM_APIS.CREATE_SEM_MODEL to create an empty model with the desired name for the copy, and use SEM_APIS.MERGE_MODELS to populate the newly created copy as the destination model.

Some common uses for this procedure include the following:

- If you have read-only access to a model that you want to modify, you can clone that model into an empty model on which you have full access, and then modify this latter model.
- If you want to consolidate multiple models, you can use this procedure as often as necessary to merge the necessary models. Merging all models beforehand and using only the merged model simplifies entailment and can improve entailment performance.

On a multi-core or multi-cpu machine, the `DOP` (degree of parallelism) option can be beneficial. See Examples for an example that uses the `DOP` option.

If the source model is large, you may want to update the optimizer statistics on the destination after the merge operation by calling the SEM_APIS.ANALYZE_MODEL procedure.

The following considerations apply to the use of this procedure:

- You must be the owner of the destination model and have SELECT privilege on the source model. If `drop_second_model=TRUE`, you must also be owner of the source model.
- This procedure is not supported on virtual models (explained in Virtual Models).
- No table constraints are allowed on the destination application table.

For information about semantic network types and options, see RDF Networks.

Examples

The following example inserts the contents of model `M1` into `M2`.

```
EXECUTE SEM_APIS.MERGE_MODELS('M1', 'M2');
```

The following example inserts the contents of model `M1` into `M2`, and it specifies a degree of parallelism of 4 (up to four parallel threads for execution of the merge operation).

```
EXECUTE SEM_APIS.MERGE_MODELS('M1', 'M2', null, null, 'DOP=4');
```

15.123 SEM_APIS.MERGE_RDF_GRAPHS

Format

```
SEM_APIS.MERGE_RDF_GRAPHS(
     source_rdf_graph      IN VARCHAR2,
     destination_rdf_graph IN VARCHAR2,
     rebuild_apptab_index  IN BOOLEAN DEFAULT TRUE,
     drop_source_rdf_graph IN BOOLEAN DEFAULT FALSE,
     options               IN VARCHAR2 DEFAULT NULL,
     network_owner         IN VARCHAR2 DEFAULT NULL,
     network_name          IN VARCHAR2 DEFAULT NULL);
```

Description

Inserts the content from a source RDF graph into a destination RDF graph, and updates the destination application table.

Parameters

source_rdf_graph
Name of the source RDF graph.

destination_model
Name of the destination RDF graph.

rebuild_apptab_index
`TRUE` causes indexes on the destination application table to be rebuilt after the graphs are merged; `FALSE` does not rebuild any indexes.

drop_source_rdf_graph
`TRUE` causes the source RDF graph (`source_rdf_graph`) to be deleted after the graphs are merged; `FALSE` (the default) does not delete the source graph.

options
A comma-delimited string of options that overrides the default behavior of the procedure. Currently, only the `DOP` (degree of parallelism) option is supported, to enable parallel execution of this procedure and to specify the degree of parallelism to be associated with the operation.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

Before you merge any RDF graphs, if you are using positional parameters, check to be sure that you are specifying the correct graphs for the first and second parameters (source RDF graph for the first, destination RDF graph for the second). This is especially important if you plan to specify `drop_source_rdf_graph=TRUE`.

If appropriate, make copies of the destination graph or both the graphs before performing the merge. To make a copy of an RDF graph, use SEM_APIS.CREATE_RDF_GRAPH to create an empty graph with the desired name for the copy, and use SEM_APIS.MERGE_RDF_GRAPHS to populate the newly created copy as the destination graph.

Some common uses for this procedure include the following:

- If you have read-only access to an RDF graph that you want to modify, you can clone that graph into an empty graph on which you have full access, and then modify this latter graph.
- If you want to consolidate multiple RDF graphs, you can use this procedure as often as necessary to merge the necessary graphs. Merging all graphs beforehand and using only the merged graph simplifies entailment and can improve entailment performance.

On a multi-core or multi-cpu machine, the `DOP` (degree of parallelism) option can be beneficial. See Examples for an example that uses the `DOP` option.

If the source graph is large, you may want to update the optimizer statistics on the destination after the merge operation by calling the SEM_APIS.ANALYZE_RDF_GRAPH procedure.

The following considerations apply to the use of this procedure:

- You must be the owner of the destination graph and have SELECT privilege on the source graph. If `drop_second_model=TRUE`, you must also be owner of the source graph.
- This procedure is not supported on RDF graph collections (explained in RDF Graph Collections).
- No table constraints are allowed on the destination application table.

For information about RDF network types and options, see RDF Networks.

Examples

The following example inserts the contents of graphl `M1` into `M2`.

```
EXECUTE SEM_APIS.MERGE_RDF_GRAPHS('M1', 'M2');
```

The following example inserts the contents of graph `M1` into `M2`, and it specifies a degree of parallelism of 4 (up to four parallel threads for execution of the merge operation).

```
EXECUTE SEM_APIS.MERGE_RDF_GRAPHS('M1', 'M2', null, null, 'DOP=4');
```

15.124 SEM_APIS.MIGRATE_DATA_TO_CURRENT

Format

```
SEM_APIS.MIGRATE_DATA_TO_CURRENT(
     options        IN VARCHAR2 DEFAULT NULL,
     network_owner  IN VARCHAR2 DEFAULT NULL,
     network_name   IN VARCHAR2 DEFAULT NULL);
```

Description

Migrates RDF data from before Oracle Database Release 21c data format to the format needed for use with RDF in the current Oracle Database release.

Parameters

options
If you specify `INS_AS_SEL=T`, the migration is performed using a bulk load operation. If you do not specify that value, then by default update operations are performed. See the Usage Notes for more information.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

You must use this procedure to migrate RDF data created using versions of Oracle Database earlier than Release 21c, as explained in Required Migration of Pre-12.2 RDF Data.

This procedure does not perform any operation on RDF data that is already in the current format. It updates the definition of RDF network triggers, views, and PL/SQL packages in the network owner's schema.

For the `options` parameter, if the amount of data to be migrated is small, the default (not specifying the parameter) probably provides adequate performance. However, for large amounts of data, specifying `INS_AS_SEL=T` can improve performance significantly.

This procedure must be run as the network owner.

Examples

The following example migrates Release 19 RDF data in a network named NET1 and owned by RDFUSER to the format for the current Oracle Database version. It performs the migration using a bulk load operation.

```
EXECUTE sem_apis.migrate_data_to_current('INS_AS_SEL=T', network_owner=>'RDFUSER',
network_name=>'NET1');
```

The following example migrates Release 19 RDF data in a network named NET1 and owned by RDFUSER to the format for the current Oracle Database version. It performs the migration using update operations (the default).

```
EXECUTE sem_apis.migrate_data_to_current(network_owner=>'RDFUSER', network_name=>'NET1');
```

15.125 SEM_APIS.MIGRATE_DATA_TO_STORAGE_V2

Format

```
SEM_APIS.MIGRATE_DATA_TO_STORAGE_V2(
     options        IN VARCHAR2 DEFAULT NULL,
     network_owner  IN VARCHAR2 DEFAULT NULL,
     network_name   IN VARCHAR2 DEFAULT NULL);
```

Description

Migrates RDF data from escaped storage form to unescaped storage form.

Parameters

options
If you specify `PARALLEL=<n>`, the migration is performed using the specified degree of parallelism. If you do not specify this option, then by default no parallel processing is used.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

It is strongly recommended that you use unescaped storage form for your RDF network, because it reduces storage cost and improves query performance, while requiring no changes to your existing applications.

This procedure must be run as the network owner.

After executing this procedure, a row with the following column values should be present in the network's RDF_PARAMETER table (described in RDF_PARAMETER Table in RDF Networks):

- Namespace: NETWORK
- Attribute: STORAGE_FORM
- Value: UNESC
- Description: Storage form setting for a RDF network.

See also Migrating from Escaped to Unescaped Storage Form.

Examples

The following example migrates an RDF network named NET1 owned by RDFUSER from escaped storage form to unescaped storage form. A degree of parallelism of 4 is used for the operation.

```
EXECUTE sem_apis.migrate_data_to_storage_v2(options=>' PARALLEL=4 ',
network_owner=>'RDFUSER', network_name=>'NET1');
```

The following example migrates an RDF network named NET1 owned by RDFUSER from escaped storage form to unescaped storage form.

```
EXECUTE sem_apis.migrate_data_to_storage_v2(network_owner=>'RDFUSER',
network_name=>'NET1');
```

15.126 SEM_APIS.MOVE_RDF_NETWORK_DATA

Format

```
SEM_APIS.MOVE_RDF_NETWORK_DATA(
     dest_schema     IN DBMS_ID,
     dest_tbs_name   IN DBMS_ID DEFAULT NULL,
     degree          IN INTEGER DEFAULT NULL,
     options         IN VARCHAR2 DEFAULT NULL,
     network_owner   IN VARCHAR2 DEFAULT NULL,
     network_name    IN VARCHAR2 DEFAULT NULL);
```

Description

Moves RDF network data from a source RDF network to a destination (staging) schema.

Parameters

dest_schema
The staging schema to which the RDF network data will be moved.

dest_tbs_name
The tablespace to use for objects created in the destination (staging) schema. If null, the default tablespace for the destination schema will be used.

degree
Degree of parallelism to use for any SQL insert or index building operations. The default is no parallel execution.

options
(Reserved for future use.)

network_owner
Owner of the source RDF network for the move operation. (See Table 1-2.)

network_name
Name of the source RDF network for the move operation. (See Table 1-2.)

Usage Notes

You must have DBA privileges to call this procedure.

For more information and examples, see Moving, Restoring, and Appending an RDF Network.

For information about RDF network types and options, see RDF Networks.

Examples

The following example moves an RDF network from the MYNET RDF network owned by RDFADMIN to the RDFEXPIMPU staging schema>

```
EXECUTE
sem_apis.move_rdf_network_data(dest_schema=>'RDFEXPIMPU',network_owner=>'RDFADMIN',networ
k_name=>'MYNET');
```

15.127 SEM_APIS.MOVE_SEM_NETWORK_DATA

Format

```
SEM_APIS.MOVE_SEM_NETWORK_DATA(
     dest_schema     IN DBMS_ID,
     dest_tbs_name   IN DBMS_ID DEFAULT NULL,
     degree          IN INTEGER DEFAULT NULL,
     options         IN VARCHAR2 DEFAULT NULL,
     network_owner   IN VARCHAR2 DEFAULT NULL,
     network_name    IN VARCHAR2 DEFAULT NULL);
```

> **Note:**
>
> This subprogram will be deprecated in a future release. It is recommended that you use the SEM_APIS.MOVE_RDF_NETWORK_DATA subprogram instead.

Description

Moves semantic network data from a source semantic network to a destination (staging) schema.

Parameters

dest_schema
The staging schema to which the semantic network data will be moved.

dest_tbs_name
The tablespace to use for objects created in the destination (staging) schema. If null, the default tablespace for the destination schema will be used.

degree
Degree of parallelism to use for any SQL insert or index building operations. The default is no parallel execution.

options
(Reserved for future use.)

network_owner
Owner of the source semantic network for the move operation. (See Table 1-2.)

network_name
Name of the source semantic network for the move operation. (See Table 1-2.)

Usage Notes

You must have DBA privileges to call this procedure.

For more information and examples, see Moving, Restoring, and Appending an RDF Network.

For information about semantic network types and options, see RDF Networks.

Examples

The following example moves a semantic network from the MYNET semantic network owned by RDFADMIN to the RDFEXPIMPU staging schema>

```
EXECUTE
sem_apis.move_sem_network_data(dest_schema=>'RDFEXPIMPU',network_owner=>'RDFADMIN',networ
k_name=>'MYNET');
```

15.128 SEM_APIS.PURGE_UNUSED_VALUES

Format

```
SEM_APIS.PURGE_UNUSED_VALUES(
     flags         IN VARCHAR2 DEFAULT NULL,
     network_owner IN VARCHAR2 DEFAULT NULL,
     network_name  IN VARCHAR2 DEFAULT NULL);
```

Description

Purges purges invalid geometry literal values from the RDF network.

Parameters

flags
An optional quoted string with one or more of the following keyword specifications:

- `MBV_METHOD=SHADOW` allows the use of a different value loading strategy that may lead to faster processing when a large number of values need to be purged.
- `PARALLEL=<integer>` allows much of the processing to be done in parallel using the specified `integer` degree of parallelism to be associated with the operation. If only `PARALLEL` is specified without a degree, a default degree will be used.
- `PUV_COMPUTE_VIDS_USED` allows use of a different strategy that may lead to faster processing when most of the values are expected to be purged.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

It is recommended that you execute this procedure after using SEM_APIS.VALIDATE_GEOMETRIES to check that all geometry literals in the specified model are valid for the provided SRID and tolerance values.

For more usage information and an extended example, see Purging Unused Values.

For information about RDF network types and options, see RDF Networks.

Examples

The following example purges unused values using a degree of parallelism of 4.

```
EXECUTE SEM_APIS.PURGE_UNUSED_VALUES(flags => 'PARALLEL=4', network_owner=>'RDFUSER',
network_name=>'NET1');
```

15.129 SEM_APIS.REFRESH_MATERIALIZED_VIEW

Format

```
SEM_APIS.REFRESH_MATERIALIZED_VIEW (
  mv_name        IN VARCHAR2,
  options        IN VARCHAR2 DEFAULT NULL,
  network_owner  IN VARCHAR2 DEFAULT NULL,
  network_name   IN VARCHAR2 DEFAULT NULL,
);
```

Description

Refreshes a materialized join view for an RDF graph stored in Oracle Database.

Parameters

mv_name
Name of the materialized view to refresh.

options
(Reserved for future use.)

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

For more information, see RDF Support for Materialized Join Views.

For information about RDF network types and options, see RDF Networks.

Examples

The following example refreshes the materialized view MVX.

```
EXECUTE SEM_APIS.REFRESH_MV_BITMAP_INDEX('MVX');
```

15.130 SEM_APIS.REFRESH_NETWORK_INDEX_INFO

Format

```
SEM_APIS.REFRESH_NETWORK_INDEX_INFO(
     options       IN VARCHAR2 DEFAULT NULL,
     network_owner IN VARCHAR2 DEFAULT NULL,
     network_name  IN VARCHAR2 DEFAULT NULL);
```

Description

Refreshes the information about RDF network indexes.

Parameters

options
(Reserved for future use)

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

This procedure updates the information in the SEM_NETWORK_INDEX_INFO view, which is described in SEM_NETWORK_INDEX_INFO View.

For information about RDF network types and options, see RDF Networks.

Examples

The following example refreshes the information about RDF network indexes.

```
EXECUTE sem_apis.refresh_network_index_info;
```

15.131 SEM_APIS.REFRESH_SEM_NETWORK_INDEX_INFO

Format

```
SEM_APIS.REFRESH_SEM_NETWORK_INDEX_INFO(
     options       IN VARCHAR2 DEFAULT NULL,
     network_owner IN VARCHAR2 DEFAULT NULL,
     network_name  IN VARCHAR2 DEFAULT NULL);
```

> **Note:**
>
> This subprogram will be deprecated in a future release. It is recommended that you use the SEM_APIS.REFRESH_NETWORK_INDEX_INFO subprogram instead.

Description

Refreshes the information about semantic network indexes.

Parameters

options
(Reserved for future use)

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

This procedure updates the information in the SEM_NETWORK_INDEX_INFO view, which is described in SEM_NETWORK_INDEX_INFO View.

For information about semantic network types and options, see RDF Networks.

Examples

The following example refreshes the information about semantic network indexes.

```
EXECUTE sem_apis.refresh_sem_network_index_info;
```

15.132 SEM_APIS.RENAME_ENTAILMENT

Format

```
SEM_APIS.RENAME_ENTAILMENT(
     old_name        IN VARCHAR2,
     new_name        IN VARCHAR2,
```

```
    network_owner  IN VARCHAR2 DEFAULT NULL,
    network_name   IN VARCHAR2 DEFAULT NULL);
```

> **Note:**
>
> This subprogram will be deprecated in a future release. It is recommended that you use the SEM_APIS.RENAME_INFERRED_GRAPH subprogram instead.

Description

Renames an entailment (rules index).

Parameters

old_name
Name of the existing entailment to be renamed.

new_name
New name for the entailment.

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

For information about semantic network types and options, see RDF Networks.

Examples

The following example renames a entailment named `OWLTST_IDX` to `MY_OWLTST_IDX`.

```
EXECUTE sem_apis.rename_entailment('owltst_idx', 'my_owltst_idx');
```

15.133 SEM_APIS.RENAME_INFERRED_GRAPH

Format

```
SEM_APIS.RENAME_INFERRED_GRAPH(
    old_name       IN VARCHAR2,
    new_name       IN VARCHAR2,
    network_owner  IN VARCHAR2 DEFAULT NULL,
    network_name   IN VARCHAR2 DEFAULT NULL);
```

Description

Renames an inferred graph (rules index).

Parameters

old_name
Name of the existing inferred graph to be renamed.

new_name
New name for the inferred graph.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

For information about RDF network types and options, see RDF Networks.

Examples

The following example renames an inferred graph named `OWLTST_IDX` to `MY_OWLTST_IDX`.

```
EXECUTE sem_apis.rename_inferred_graph('owltst_idx', 'my_owltst_idx');
```

15.134 SEM_APIS.RENAME_MODEL

Format

```
SEM_APIS.RENAME_MODEL(
     old_name        IN VARCHAR2,
     new_name        IN VARCHAR2,
     network_owner   IN VARCHAR2 DEFAULT NULL,
     network_name    IN VARCHAR2 DEFAULT NULL);
```

> **Note:**
>
> This subprogram will be deprecated in a future release. It is recommended that you use the SEM_APIS.RENAME_RDF_GRAPH subprogram instead.

Description

Renames a model.

Parameters

old_name
Name of the existing model to be renamed.

new_name
New name for the model.

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

The following considerations apply to the use of this procedure:

- You must be the owner of the existing model.

- This procedure is not supported on virtual models (explained in Virtual Models).

Contrast this procedure with SEM_APIS.SWAP_NAMES, which swaps (exchanges) the names of two existing models.

For information about semantic network types and options, see RDF Networks.

Examples

The following example renames a model named `MODEL1` to `MODEL2`.

```
EXECUTE sem_apis.rename_model('model1', 'model2');
```

15.135 SEM_APIS.RENAME_RDF_GRAPH

Format

```
SEM_APIS.RENAME_RDF_GRAPH(
     old_name       IN VARCHAR2,
     new_name       IN VARCHAR2,
     network_owner  IN VARCHAR2 DEFAULT NULL,
     network_name   IN VARCHAR2 DEFAULT NULL);
```

Description

Renames an RDF graph.

Parameters

old_name
Name of the existing RDF graph to be renamed.

new_name
New name for the RDF graph.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

The following considerations apply to the use of this procedure:

- You must be the owner of the existing RDF graph.
- This procedure is not supported on RDF graph collections (explained in RDF Graph Collections).

Contrast this procedure with SEM_APIS.SWAP_NAMES, which swaps (exchanges) the names of two existing graphs.

For information about RDF network types and options, see RDF Networks.

Examples

The following example renames a graph named `G1` to `G2`.

```
EXECUTE sem_apis.rename_rdf_graph('g1', 'g2');
```

15.136 SEM_APIS.RES2VID

Format

```
SEM_APIS.RES2VID(
     vTab       IN VARCHAR2,
     uri        IN VARCHAR2,
     lt         IN VARCHAR2 DEFAULT NULL,
     lang       IN VARCHAR2 DEFAULT NULL,
     lval       IN CLOB DEFAULT NULL,
     vtyp       IN VARCHAR2 DEFAULT NULL,
     max_vc_len IN NUMBER DEFAULT 4000
     ) RETURN NUMBER;
```

Description

Returns the VALUE_ID for the canonical version of an RDF term, or NULL if the term does not exist in the values table.

Parameters

vTab
Values table to query for the VALUE_ID value. (Usually RDF_VALUE$)

uri
Prefix value of the RDF term.

lt
Data type URI of a types literal to look up. Do not include the enclosing angle brackets (‘<’ and ‘>’).

lang
Language tag of a language tagged literal to look up.

lval
The plain literal portion of a long literal to look up.

vtyp
The type of value:

- **PL:** Plain literal
- **TL:** Typed literal
- **UR:** URI/IRI
- **BN:** Blank node
- **PL@ language:** Tagged literal

The value type is determined automatically if this parameter is NULL.

max_vc_len
The maximum allowed length of a VARCHAR RDF term: 32767 or 4000 (the default).

Usage Notes

For information about the components of an RDF term stored in the RDF_VALUE$ table, see RDF Metadata Tables and Views..

Examples

The following example returns VALUE_ID values for the canonical versions of RDF terms. Comments before each SQL statement describe the purpose of the statement.

```
-- Look up the VALUE_ID for the RDF term <http://www.example.com/a>.
SELECT  sem_apis.res2vid('RDF_VALUE$','<http://www.example.com/a>') FROM DUAL;

-- Look up the VALUE_ID for the RDF term "abc".
SELECT sem_apis.res2vid('RDF_VALUE$','"abc"') FROM DUAL;

-- Look up the VALUE_ID for the RDF term "10"^^<http://www.w3.org/2001/
XMLSchema#decimal>.
SELECT sem_apis.res2vid('RDF_VALUE$','"10"','http://www.w3.org/2001/XMLSchema#decimal')
FROM DUAL;

-- Look up the VALUE_ID for the RDF term "abc"@en.
SELECT sem_apis.res2vid('RDF_VALUE$','"abc"',lang=>'en') FROM DUAL;

-- Look up the VALUE_ID for the long literal RDF term '"a CLOB literal"'.
SELECT sem_apis.res2vid('RDF_VALUE$',null,lval=>'"a CLOB literal"') FROM DUAL;
```

15.137 SEM_APIS.RESTORE_RDF_NETWORK_DATA

Format

```
SEM_APIS.RESTORE_RDF_NETWORK_DATA(
     from_schema    DBMS_ID,
     degree         INTEGER DEFAULT NULL,
     options        VARCHAR2 DEFAULT NULL,
     network_owner  IN VARCHAR2 DEFAULT NULL,
     network_name   IN VARCHAR2 DEFAULT NULL);
```

Description

Restores moved RDF network data from a staging schema back into a source RDF network.

Parameters

from_schema
The staging schema that contains moved RDF network data to be restored.

degree
Degree of parallelism to use for any SQL insert or index building operations. The default is no parallel execution.

options
String specifying any options to use during the append operation. Supported options are:

- PURGE=T – drop all remaining RDF network data in the staging schema after the append operation completes.

network_owner
Owner of the destination RDF network for the restore operation. (See Table 1-2.)

network_name
Name of the destination RDF network for the restore operation. (See Table 1-2.)

Usage Notes

Partition exchange operations rather than SQL INSERT statements are used to move most of the data during the append operation, so the staging schema will no longer contain complete RDF network data after the restore operation is complete.

Moved RDF network data can only be restored into the original source RDF network from which it was moved.

You must have DBA privileges to call this procedure.

For more information, see Moving, Restoring, and Appending an RDF Network.

For information about RDF network types and options, see RDF Networks.

Examples

The following example restores an RDF network from the RDFEXPIMPU staging schema into the MYNET RDF network owned by RDFADMIN.

```
EXECUTE
sem_apis.restore_rdf_network_data(from_schema=>'RDFEXPIMPU',network_owner=>'RDFADMIN',net
work_name=>'MYNET');
```

15.138 SEM_APIS.RESTORE_SEM_NETWORK_DATA

Format

```
SEM_APIS.RESTORE_SEM_NETWORK_DATA(
     from_schema    DBMS_ID,
     degree         INTEGER DEFAULT NULL,
     options        VARCHAR2 DEFAULT NULL,
     network_owner  IN VARCHAR2 DEFAULT NULL,
     network_name   IN VARCHAR2 DEFAULT NULL);
```

> **Note:**
>
> This subprogram will be deprecated in a future release. It is recommended that you use the SEM_APIS.RESTORE_RDF_NETWORK_DATA subprogram instead.

Description

Restores moved semantic network data from a staging schema back into a source semantic network.

Parameters

from_schema
The staging schema that contains moved semantic network data to be restored.

degree
Degree of parallelism to use for any SQL insert or index building operations. The default is no parallel execution.

options
String specifying any options to use during the append operation. Supported options are:

- PURGE=T – drop all remaining semantic network data in the staging schema after the append operation completes.

network_owner
Owner of the destination semantic network for the restore operation. (See Table 1-2.)

network_name
Name of the destination semantic network for the restore operation. (See Table 1-2.)

Usage Notes

Partition exchange operations rather than SQL INSERT statements are used to move most of the data during the append operation, so the staging schema will no longer contain complete semantic network data after the restore operation is complete.

Moved semantic network data can only be restored into the original source semantic network from which it was moved.

You must have DBA privileges to call this procedure.

For more information, see Moving, Restoring, and Appending an RDF Network.

For information about semantic network types and options, see RDF Networks.

Examples

The following example restores a semantic network from the RDFEXPIMPU staging schema into the MYNET semantic network owned by RDFADMIN.

```
EXECUTE 
sem_apis.restore_sem_network_data(from_schema=>'RDFEXPIMPU',network_owner=>'RDFADMIN',net
work_name=>'MYNET');
```

15.139 SEM_APIS.REVOKE_MODEL_ACCESS_PRIV

Format

```
SEM_APIS.REVOKE_MODEL_ACCESS_PRIV(
     model_name      IN VARCHAR2,
     user_name       IN VARCHAR2,
     privilege       IN VARCHAR2,
     user_view       IN VARCHAR2 DEFAULT NULL,
     options         IN VARCHAR2 DEFAULT NULL,
     network_owner   IN VARCHAR2 DEFAULT NULL,
     network_name    IN VARCHAR2 DEFAULT NULL);
```

> **Note:**
>
> This subprogram will be deprecated in a future release. It is recommended that you use the SEM_APIS.REVOKE_RDF_GRAPH_ACCESS_PRIV subprogram instead.

Description

Revokes access privilege on a model or entailment.

Parameters

model_name
Name of the model.

user_name
Database user that is recipient of this privilege.

privilege
Specifies the type of privilege that is granted. Currently allowed values include the following:

- `QUERY`: Query the model using SPARQL
- `SELECT`, `READ`: Retrieve model content using SQL. The source for the content is the `RDFT_<model>` view in the network owner's schema or the view name, if any, specified for the `user_view` parameter.
- `INSERT`, `UPDATE`, `DELETE`: Perform SPARQL Update (DML) operations on the model or SQL DML operations. For SQL DML, the target object is the `RDFT_<model>` view in the network owner's schema.

> **Note:**
>
> `QUERY` is the only valid choice if the model is not a regular model (that is, not created using `sem_apis.create_sem_model`).

user_view
Applicable to schema-private network only. If a view was created on the `RDFT_<model>` view at model creation time using `sem_apis.create_sem_model` or later, privilege is revoked on that view.

options
If user specifies the word `ENTAILMENT` as part of the string value, then the specified `model_name` is taken as the name of an entailment (rules index). (Additional words or phrases may be allowed in future.)

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

This does not affect the recipient's query-only or full access to the semantic network (which guarantees access to dictionary tables, but not individual models). This operation revokes access to the specified model only.

Examples

The following example revokes privilege from database user `USER1` for use of SPARQL query against a semantic technology model named `articles` in the schema-private network `NET1` owned by database user `RDFUSER`. (This example refers to the model described in Example 1-129.)

```
EXECUTE SEM_APIS.REVOKE_MODEL_ACCESS_PRIV('articles', 'USER1', 'QUERY',
network_owner=>'RDFUSER', network_name=>'NET1');
```

15.140 SEM_APIS.REVOKE_MODEL_ACCESS_PRIVS

Format

```
SEM_APIS.REVOKE_MODEL_ACCESS_PRIVS(
     model_name       IN VARCHAR2,
     user_name        IN VARCHAR2,
     priv_list        IN VARCHAR2,
     user_view        IN VARCHAR2 DEFAULT NULL,
     options          IN VARCHAR2 DEFAULT NULL,
     network_owner    IN VARCHAR2 DEFAULT NULL,
     network_name     IN VARCHAR2 DEFAULT NULL);
```

> **Note:**
>
> This subprogram will be deprecated in a future release. It is recommended that you use the SEM_APIS.REVOKE_RDF_GRAPH_ACCESS_PRIVS subprogram instead.

Description

Revokes access privileges on a model or entailment.

Parameters

model_name
Name of the model.

user_name
Database user that is recipient of this privilege.

priv_list
Specifies the type of privilege that is granted. Currently allowed values include the following:

- `QUERY`: Query the model using SPARQL
- `SELECT`, `READ`: Retrieve model content using SQL. The source for the content is the RDFT_<model> view in the network owner's schema or the view name, if any, specified for the `user_view` parameter.
- `INSERT`, `UPDATE`, `DELETE`: Perform SPARQL Update (DML) operations on the model or SQL DML operations. For SQL DML, the target object is the RDFT_<model> view in the network owner's schema.

> **Note:**
>
> `QUERY` is the only valid choice if the model is not a regular model (that is, not created using `sem_apis.create_sem_model`).

user_view
Applicable to schema-private network only. If a view was created on the `RDFT_<model>` view at model creation time using `sem_apis.create_sem_model` or later, privileges are revoked on that view.

options
If user specifies the word `ENTAILMENT` as part of the string value, then the specified `model_name` is taken as the name of an entailment (rules index). (Additional words or phrases may be allowed in future.)

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

This does not affect the recipient's query-only or full access to the semantic network (which guarantees access to dictionary tables, but not individual models). This operation revokes access to the specified model only.

Examples

The following example revokes privilege from database user `USER1` for performing DML operations against a semantic technology model named `articles` in the schema-private network `NET1` owned by database user `RDFUSER`. (This example refers to the model described in Example 1-129.)

```
EXECUTE SEM_APIS.REVOKE_MODEL_ACCESS_PRIVS('articles', 'USER1',
sys.odcivarchar2list('INSERT','UPDATE','DELETE'), network_owner=>'RDFUSER',
network_name=>'NET1');
```

15.141 SEM_APIS.REVOKE_NETWORK_ACCESS_PRIVS

Format

```
SEM_APIS.REVOKE_NETWORK_ACCESS_PRIVS(
     network_owner  IN VARCHAR2,
     network_name   IN VARCHAR2,
     network_user   IN VARCHAR2,
     options        IN VARCHAR2 default NULL);
```

Description

Revokes access privileges from a database user other than the owner of a schema-private RDF network.

Parameters

network_owner
Owner of the RDF network. (Cannot be MDSYS.)

network_name
Name of the RDF network. (Must be a schema-private network.)

network_user
Database user (other than the network owner) from which to revoke access privileges to the network.

options
String specifying options for access using the form *OPTION_NAME=option_value*.
If `CASCADE=T` is specified, any RDF objects owned by the database user will be dropped as part of this operation.

Usage Notes

You must have DBA privileges or be the owner of the specified network to call this procedure.

If the database user (`network_user`) owns any RDF objects in the schema-private network and if `CASCADE=T` is *not* specified, an error will be raised.

For information about RDF network types and options, see RDF Networks.

Examples

The following example revokes full access on the `mynet1` network owned by `scott` from `rdfuser1`.

```
EXECUTE SEM_APIS.REVOKE_NETWORK_ACCESS_PRIVS('scott','mynet1','rdfuser1');
```

15.142 SEM_APIS.REVOKE_NETWORK_SHARING_PRIVS

Format

```
SEM_APIS.REVOKE_NETWORK_SHARING_PRIVS(
     network_owner  IN VARCHAR2,
     options        IN VARCHAR2 default NULL);
```

> **Note:**
>
> You can skip the `REVOKE_NETWORK_SHARING_PRIVS` procedure if your Oracle Database version is 23.4.

Description

Revokes from a database user the privileges required for sharing, with other database users, any schema-private networks owned (currently or in the future) by the database user

Parameters

network_owner
Owner of the network. (Cannot be MDSYS.)

options
(Reserved for future use)

Usage Notes

You must have DBA privileges to call this procedure.

If the database user owns at least one schema-private network that has sharing enabled, an exception will be raised. (The user must first disable sharing of any such networks.)

For information about RDF network types and options, see RDF Networks.

Examples

The following example revokes from database user `scott` the privileges for sharing any schema-private networks that this user owns or will own.

```
EXECUTE SEM_APIS.REVOKE_NETWORK_SHARING_PRIVS('scott');
```

15.143 SEM_APIS.REVOKE_RDF_GRAPH_ACCESS_PRIV

Format

```
SEM_APIS.REVOKE_RDF_GRAPH_ACCESS_PRIV(
     rdf_graph_name  IN VARCHAR2,
     user_name       IN VARCHAR2,
     privilege       IN VARCHAR2,
     user_view       IN VARCHAR2 DEFAULT NULL,
     options         IN VARCHAR2 DEFAULT NULL,
     network_owner   IN VARCHAR2 DEFAULT NULL,
     network_name    IN VARCHAR2 DEFAULT NULL);
```

Description

Revokes access privilege on an RDF graph or inferred graph.

Parameters

rdf_graph_name
Name of the RDF graph.

user_name
Database user that is recipient of this privilege.

privilege
Specifies the type of privilege that is granted. Currently allowed values include the following:

- `QUERY`: Query the RDF graph using SPARQL
- `SELECT`, `READ`: Retrieve RDF graph content using SQL. The source for the content is the `RDFT_<model>` view in the network owner's schema or the view name, if any, specified for the `user_view` parameter.
- `INSERT`, `UPDATE`, `DELETE`: Perform SPARQL Update (DML) operations on the RDF graph or SQL DML operations. For SQL DML, the target object is the `RDFT_<rdf_graph_name>` view in the network owner's schema.

> **Note:**
>
> `QUERY` is the only valid choice if the RDF graph is not a regular graph (that is, not created using `sem_apis.create_rdf_graph`).

user_view
Applicable to schema-private network only. If a view was created on the `RDFT_<rdf_graph_name>` view at RDF graph creation time using `sem_apis.create_rdf_graph` or later, privilege is revoked on that view.

options
If user specifies the word `ENTAILMENT` as part of the string value, then the specified `rdf_graph_name` is taken as the name of an inferred graph (rules index). (Additional words or phrases may be allowed in future.)

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

This does not affect the recipient's query-only or full access to the RDF network (which guarantees access to dictionary tables, but not individual RDF graphs). This operation revokes access to the specified graph only.

Examples

The following example revokes privilege from database user `USER1` for use of SPARQL query against an RDF graph named `articles` in the schema-private network `NET1` owned by database user `RDFUSER`. (This example refers to the RDF graph described in Example 1-129.)

```
EXECUTE SEM_APIS.REVOKE_RDF_GRAPH_ACCESS_PRIV('articles', 'USER1', 'QUERY',
network_owner=>'RDFUSER', network_name=>'NET1');
```

15.144 SEM_APIS.REVOKE_RDF_GRAPH_ACCESS_PRIVS

Format

```
SEM_APIS.REVOKE_RDF_GRAPH_ACCESS_PRIVS(
     rdf_graph_name  IN VARCHAR2,
     user_name       IN VARCHAR2,
     priv_list       IN VARCHAR2,
     user_view       IN VARCHAR2 DEFAULT NULL,
     options         IN VARCHAR2 DEFAULT NULL,
     network_owner   IN VARCHAR2 DEFAULT NULL,
     network_name    IN VARCHAR2 DEFAULT NULL);
```

Description

Revokes access privileges on an RDF graph or inferred graph.

Parameters

rdf_graph_name
Name of the RDF graph.

user_name
Database user that is recipient of this privilege.

priv_list
Specifies the type of privilege that is granted. Currently allowed values include the following:

- `QUERY`: Query the RDF graph using SPARQL

- `SELECT`, `READ`: Retrieve RDF graph content using SQL. The source for the content is the `RDFT_<rdf_graph_name>` view in the network owner's schema or the view name, if any, specified for the `user_view` parameter.
- `INSERT`, `UPDATE`, `DELETE`: Perform SPARQL Update (DML) operations on the RDF graph or SQL DML operations. For SQL DML, the target object is the `RDFT_<rdf_graph_name>` view in the network owner's schema.

> **Note:**
>
> `QUERY` is the only valid choice if the RDF graph is not a regular RDF graph (that is, not created using `sem_apis.create_rdf_graph`).

user_view
Applicable to schema-private network only. If a view was created on the `RDFT_<rdf_graph_name>` view at RDF graph creation time using `sem_apis.create_rdf_graph` or later, privileges are revoked on that view.

options
If user specifies the word `ENTAILMENT` as part of the string value, then the specified `rdf_graph_name` is taken as the name of an inferred graph (rules index). (Additional words or phrases may be allowed in future.)

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

This does not affect the recipient's query-only or full access to the RDF network (which guarantees access to dictionary tables, but not individual RDF graphs). This operation revokes access to the specified RDF graph only.

Examples

The following example revokes privilege from database user `USER1` for performing DML operations against a semantic technology RDF graph named `articles` in the schema-private network `NET1` owned by database user `RDFUSER`. (This example refers to the RDF graph described in Example 1-129.)

```
EXECUTE SEM_APIS.REVOKE_RDF_GRAPH_ACCESS_PRIVS('articles', 'USER1',
sys.odcivarchar2list('INSERT','UPDATE','DELETE'), network_owner=>'RDFUSER',
network_name=>'NET1');
```

15.145 SEM_APIS.SEM_SQL_COMPILE

Format

```
SEM_APIS.SEM_SQL_COMPILE;
```

Description

Compiles SQL inserted into `RDF$S2S_SQL$` table to be used by SEM_SQL SQL Macro.

Parameters

Usage Notes

Examples

The following example compiles SQL inserted into `RDF$S2S_SQL$` table.

```
INSERT INTO RDF$S2S_SQL$ SELECT s2s_sql FROM sql_tab WHERE id=1;

EXECUTE SEM_APIS.SEM_SQL_COMPILE;

SELECT count(s), count(o) FROM SEM_SQL();
```

15.146 SEM_APIS.SET_ENTAILMENT_STATS

Format

```
SEM_APIS.SET_ENTAILMENT_STATS (
     entailment_name IN VARCHAR2,
     numrows         IN NUMBER DEFAULT NULL,
     numblks         IN NUMBER DEFAULT NULL,
     avgrlen         IN NUMBER DEFAULT NULL,
     flags           IN NUMBER DEFAULT NULL,
     no_invalidate   IN BOOLEAN DEFAULT DBMS_STATS.AUTO_INVALIDATE,
     cachedblk       IN NUMBER DEFAULT NULL,
     cachehit        IN NUMBER DEFAULT NULL,
     force           IN BOOLEAN DEFAULT FALSE,
     network_owner   IN VARCHAR2 DEFAULT NULL,
     network_name    IN VARCHAR2 DEFAULT NULL);
```

Description

Sets statistics for a specified entailment.

Parameters

entailment_name
Name of the entailment.

(other parameters)
See the parameter explanations for the DBMS_STATS.SET_TABLE_STATS procedure in *Oracle Database PL/SQL Packages and Types Reference*, although `force` here applies to entailment statistics.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

See the information about the DBMS_STATS package in*Oracle Database PL/SQL Packages and Types Reference*.

See also Managing Statistics for Semantic Models and the Semantic Network.

For information about RDF network types and options, see RDF Networks.

Examples

The following example sets statistics for an entailment named `OWLTST_IDX`.

```
EXECUTE SEM_APIS.SET_ENTAILMENT_STATS('owltst_idx', numrows => 100);
```

15.147 SEM_APIS.SET_MODEL_STATS

Format

```
SEM_APIS.SET_MODEL_STATS (
     model_name     IN VARCHAR2,
     numrows        IN NUMBER DEFAULT NULL,
     numblks        IN NUMBER DEFAULT NULL,
     avgrlen        IN NUMBER DEFAULT NULL,
     flags          IN NUMBER DEFAULT NULL,
     no_invalidate  IN BOOLEAN DEFAULT DBMS_STATS.AUTO_INVALIDATE,
     cachedblk      IN NUMBER DEFAULT NULL,
     cachehit       IN NUMBER DEFAULT NULL,
     force          IN BOOLEAN DEFAULT FALSE,
     network_owner  IN VARCHAR2 DEFAULT NULL,
     network_name   IN VARCHAR2 DEFAULT NULL);
```

Description

Sets statistics for a specified model.

Parameters

model_name
Name of the model.

(other parameters)
See the parameter explanations for the DBMS_STATS.DELETE_TABLE_STATS procedure in *Oracle Database PL/SQL Packages and Types Reference*, although `force` here applies to model statistics.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

See the information about the DBMS_STATS package in*Oracle Database PL/SQL Packages and Types Reference*.

See also Managing Statistics for Semantic Models and the Semantic Network.

For information about RDF network types and options, see RDF Networks.

Examples

The following example sets statistics for a model named `FAMILY`.

```
EXECUTE SEM_APIS.SET_MODEL_STATS('family', numrows => 100);
```

15.148 SEM_APIS.SPARQL_TO_SQL

Format

```
SEM_APIS.SPARQL_TO_SQL(
    sparql_query  IN CLOB,
    models        IN RDF_MODELS DEFAULT NULL,
    rulebases     IN RDF_RULEBASES DEFAULT NULL,
    aliases       IN RDF_ALIASES DEFAULT NULL,
    index_status  IN VARCHAR2 DEFAULT NULL,
    options       IN VARCHAR2 DEFAULT NULL
    graphs        IN RDF_GRAPHS DEFAULT NULL,
    named_graphs  IN RDF_GRAPHS DEFAULT NULL,
    network_owner IN VARCHAR2 DEFAULT NULL,
    network_name  IN VARCHAR2 DEFAULT NULL) RETURN CLOB;
```

Description

Translates a SPARQL query into a SQL query string that can be executed by an application program.

Parameters

sparql_query
A string literal with one or more triple patterns, usually containing variables.

models
The model or models to use.

rulebases
One or more rulebases whose rules are to be applied to the query

aliases
One or more namespaces to be used for expansion of qualified names in the query pattern.

index_status
The status of the relevant entailment for this query.

options
Options that can affect the results of queries.

graphs
The set of named graphs from which to construct the default graph for the query.

named_graphs
The set of named graphs that can be matched by a GRAPH clause.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

Before using this procedure, be sure you understand the material in Using the SEM_APIS.SPARQL_TO_SQL Function to Query RDF Data.

For information about RDF network types and options, see RDF Networks.

Examples

The following example translates a SPARQL query into a SQL query string.

```
DECLARE
  sparql_stmt clob;
  sql_stmt    clob;
BEGIN
  sparql_stmt := '{?x :grandParentOf ?y . ?x rdf:type :Male}';
  sql_stmt := sem_apis.sparql_to_sql(
                sparql_stmt,
                sem_models('family'),
                SEM_Rulebases('RDFS','family_rb'),
                SEM_ALIASES(SEM_ALIAS('','http://www.example.org/family/')),
                null);
  execute immediate
    'create table gf_table as
     select x grandfather, y grandchild from('|| sql_stmt || ')';
END;
/
```

15.149 SEM_APIS.SWAP_NAMES

Format

```
SEM_APIS.SWAP_NAMES(
     model1         IN VARCHAR2,
     model2         IN VARCHAR2,
     network_owner  IN VARCHAR2 DEFAULT NULL,
     network_name   IN VARCHAR2 DEFAULT NULL);
```

Description

Swaps (exchanges) the names of two existing models.

Parameters

model1
Name of a model.

model2
Name of another model.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

As a result of this procedure, the name of model `model1` is changed to the (old) name of `model2`, and the name of model `model2` is changed to the (old) name of `model1`.

The order of the names does not affect the result. For example, you could specify `TEST` for `model1` and `PRODUCTION` for `model2`, or `PRODUCTION` for `model1` and `TEST` for `model2`, and the result will be the same.

Contrast this procedure with SEM_APIS.RENAME_MODEL, which renames an existing model.

For information about semantic network types and options, see RDF Networks.

Examples

The following example changes the name of the (old) TEST model to PRODUCTION, and the name of the (old) PRODUCTION model to TEST.

```
EXECUTE sem_apis.swap_names('test', 'production');
```

15.150 SEM_APIS.TRUNCATE_SEM_MODEL

Format

```
SEM_APIS.TRUNCATE_SEM_MODEL(
     model_name    IN VARCHAR2,
     options       IN VARCHAR2 DEFAULT NULL,
     network_owner IN VARCHAR2 DEFAULT NULL,
     network_name  IN VARCHAR2 DEFAULT NULL);
```

> **Note:**
>
> This subprogram will be deprecated in a future release. It is recommended that you use the SEM_APIS.TRUNCATE_RDF_GRAPH subprogram instead.

Description

Truncates a semantic technology model.

Parameters

model_name
Name of the model.

options
(Reserved for future use)

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

This procedure removes all triples and quads from the specified semantic model and is the only supported way to truncate a model.

To delete a model, use the SEM_APIS.DROP_SEM_MODEL procedure.

For information about semantic network types and options, see RDF Networks.

Examples

The following example truncates a semantic technology model named `articles`. (This example refers to the model described in Example 1-129.)

```
EXECUTE SEM_APIS.TRUNCATE_SEM_MODEL('articles', NULL, network_owner=>'RDFUSER',
network_name=>'NET1');
```

15.151 SEM_APIS.TRUNCATE_RDF_GRAPH

Format

```
SEM_APIS.TRUNCATE_RDF_GRAPH(
     rdf_graph_name IN VARCHAR2,
     options        IN VARCHAR2 DEFAULT NULL,
     network_owner  IN VARCHAR2 DEFAULT NULL,
     network_name   IN VARCHAR2 DEFAULT NULL);
```

Description

Truncates an RDF graph.

Parameters

rdf_graph_name
Name of the RDF graph.

options
(Reserved for future use)

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

This procedure removes all triples and quads from the specified RDF graph and is the only supported way to truncate a graph.

To delete an RDF graph, use the SEM_APIS.DROP_RDF_GRAPH procedure.

For information about RDF network types and options, see RDF Networks.

Examples

The following example truncates an RDF graph named `articles`. (This example refers to the graph described in Example 1-129.)

```
EXECUTE SEM_APIS.TRUNCATE_RDF_GRAPH('articles', NULL, network_owner=>'RDFUSER',
network_name=>'NET1');
```

15.152 SEM_APIS.UNESCAPE_CLOB_TERM

Format

```
SEM_APIS.UNESCAPE_CLOB_TERM(
     term         IN CLOB CHARACTER SET ANY_CS,
     options      IN VARCHAR2 DEFAULT NULL,
     max_vc_len   IN NUMBER DEFAULT 4000
     ) RETURN CLOB CHARACTER SET val%CHARSET;
```

Description

Returns the input RDF term with special characters and non-ASCII characters unescaped as specified by the W3C N-Triples format (`http://www.w3.org/TR/rdf-testcases/#ntriples`).

Parameters

term
The RDF term to unescape.

options
Reserved for future use.

max_vc_len
The maximum allowed length of a VARCHAR RDF term - 32767 or 4000 (the default).

Usage Notes

For information about using the `DO_UNESCAPE` keyword in the `options` parameter of the SEM_MATCH table function, see Using the SEM_MATCH Table Function to Query Semantic Data.

Examples

The following example unescapes an input RDF term containing TAB and NEWLINE characters.

```
SEM_APIS.UNESCAPE_CLOB_TERM('"abc\tdef\nhij"^^<http://www.w3.org/2001/XMLSchema#string>')
  FROM DUAL;
```

15.153 SEM_APIS.UNESCAPE_CLOB_VALUE

Format

```
SEM_APIS.UNESCAPE_CLOB_VALUE(
     val           IN  CLOB CHARACTER SET ANY_CS,
     start_offset  IN NUMBER DEFAULT 1,
     end_offset    IN NUMBER DEFAULT 0,
     include_start IN NUMBER DEFAULT 0,
     options       IN VARCHAR2 DEFAULT NULL,
     max_vc_len    IN NUMBER DEFAULT 4000
     ) RETURN VARCHAR2 CHARACTER SET val%CHARSET;
```

Description

Returns the input CLOB value with special characters and non-ASCII characters unescaped as specified by the W3C N-Triples format (`http://www.w3.org/TR/rdf-testcases/#ntriples`).

Parameters

val
The CLOB text to unescape.

start_offset
The offset in `val` from which to start character unescaping. The default (1) causes escaping to start at the first character of `val`.

end_offset
The offset in `val` from which to end character unescaping. The default (0) causes escaping to continue through the end of `val`.

include_start
Set to 1 if the characters in val from 1 to `start_offset` should be prefixed (prepended) to the return value. Otherwise, no such characters will be prefixed to the return value.

options
Reserved for future use.

max_vc_len
The maximum allowed length of a VARCHAR RDF term - 32767 or 4000 (the default).

Usage Notes

For information about using the `DO_UNESCAPE` keyword in the `options` parameter of the SEM_MATCH table function, see Using the SEM_MATCH Table Function to Query Semantic Data.

Examples

The following example unescapes an input character string containing TAB and NEWLINE characters.

```
SELECT SEM_APIS.UNESCAPE_CLOB_VALUE('abc\tdef\nhij')
  FROM DUAL;
```

15.154 SEM_APIS.UNESCAPE_RDF_TERM

Format

```
SEM_APIS.UNESCAPE_RDF_TERM(
     term       IN VARCHAR2 CHARACTER SET ANY_CS,
     options    IN VARCHAR2 DEFAULT NULL,
     max_vc_len IN NUMBER DEFAULT 4000
     ) RETURN VARCHAR2 CHARACTER SET val%CHARSET;
```

Description

Returns the input RDF term with special characters and non-ASCII characters unescaped as specified by the W3C N-Triples format (`http://www.w3.org/TR/rdf-testcases/#ntriples`).

Parameters

term
The RDF term to unescape.

options
Reserved for future use.

max_vc_len
The maximum allowed length of a VARCHAR RDF term - 32767 or 4000 (the default).

Usage Notes

For information about using the `DO_UNESCAPE` keyword in the `options` parameter of the SEM_MATCH table function, see Using the SEM_MATCH Table Function to Query Semantic Data.

Examples

The following example unescapes an input RDF term containing TAB and NEWLINE characters.

```
SELECT SEM_APIS.UNESCAPE_RDF_TERM('"abc\tdef\nhij"^^<http://www.w3.org/2001/
XMLSchema#string>')
  FROM DUAL;
```

15.155 SEM_APIS.UNESCAPE_RDF_VALUE

Format

```
SEM_APIS.UNESCAPE_RDF_VALUE(
     val         IN VARCHAR2 CHARACTER SET ANY_CS,
     options     IN VARCHAR2 DEFAULT NULL,
     max_vc_len  IN NUMBER DEFAULT 4000
     ) RETURN VARCHAR2 CHARACTER SET val%CHARSET;
```

Description

Returns the input CLOB value with special characters and non-ASCII characters unescaped as specified by the W3C N-Triples format (`http://www.w3.org/TR/rdf-testcases/#ntriples`).

Parameters

val
The text to unescape.

options
Reserved for future use.

max_vc_len
The maximum allowed length of a VARCHAR RDF term - 32767 or 4000 (the default).

Usage Notes

For information about using the `DO_UNESCAPE` keyword in the `options` parameter of the SEM_MATCH table function, see Using the SEM_MATCH Table Function to Query Semantic Data.

Examples

The following example unescapes an input character string containing TAB and NEWLINE characters.

```
SELECT SEM_APIS.UNESCAPE_RDF_VALUE('abc\tdef\nhij')
  FROM DUAL;
```

15.156 SEM_APIS.UPDATE_MODEL

Format

```
SEM_APIS.UPDATE_MODEL(
     apply_model          IN VARCHAR2,
     update_stmt          IN CLOB,
     match_models         IN RDF_MODELS DEFAULT NULL,
     match_rulebases      IN RDF_RULEBASES DEFAULT NULL,
     match_index_status   IN VARCHAR2 DEFAULT NULL,
     match_options        IN VARCHAR2 DEFAULT NULL,
     options              IN VARCHAR2 DEFAULT NULL,
     network_owner        IN VARCHAR2 DEFAULT NULL,
     network_name         IN VARCHAR2 DEFAULT NULL);
```

> **Note:**
>
> This subprogram will be deprecated in a future release. It is recommended that you use the SEM_APIS.UPDATE_RDF_GRAPH subprogram instead.

Description

Executes a SPARQL Update statement on a semantic model.

Parameters

apply_model
Name of the RDF model to be updated. This is the name specified when the model was created using the SEM_APIS.CREATE_SEM_MODEL procedure.
It cannot be a virtual model (see RDF Graph Collections) or an RDF view).

update_stmt
One or more SPARQL Update commands to be executed on the `apply_model` model. Use the semicolon (;) to separate commands.

match_models
A list of models that forms the SPARQL data set to query for graph pattern matching during a SPARQL Update operation (INSERT WHERE, DELETE WHERE, COPY, MOVE, ADD). Can include virtual models and/or RDF views If this parameter is not specified, the `apply_model` model is used.

match_rulebases
A list of rulebases to use with `match_models` to provide an entailment that generates additional triples or quads to use for graph pattern matching during a SPARQL Update operation.

match_index_status
The desired status for any entailments used for graph pattern matching during a SPARQL Update operation.

match_options
String specifying hints to influence graph pattern matching during a SPARQL Update operation. The set of hints that can be used here is identical to those that can be used in the `options` parameter of SEM_MATCH.

options
String specifying hints that affect SPARQL operations. See the Usage Notes for a list of available options.

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

Before using this procedure, be sure you understand the material in Support for SPARQL Update Operations on an RDF Graph.

The `options` parameter can specify one or more of the following options:

- **APP_TAB_IDX={INDEX_NAME}** uses an INDEX optimizer hint for INDEX_NAME when doing DML operations on the application table.
- **APPEND** uses the SQL APPEND hint with DML operations.
- **AUTOCOMMIT=F** avoids starting and committing a transaction for each SEM_APIS.UPDATE_MODEL call. Instead, this option gives transaction control to the caller. Each SEM_APIS.UPDATE_MODEL call will execute as part of a main transaction that is started, committed, or rolled back by the caller.
- **BULK_OPTIONS={OPTIONS_STRING}** uses OPTIONS_STRING as the `flags` parameter when calling SEM_APIS.BULK_LOAD_FROM_STAGING_TABLE.
- **CLOB_UPDATE_SUPPORT=T** turns on CLOB functionality.
- **DEL_AS_INS=T** performs a large delete operation by inserting all data that should remain after the delete operation instead of doing deletions. This option may significantly improve the performance of large delete operations.
- **DYNAMIC_SAMPLING(n)** uses DYNAMIC_SAMPLING(n) SQL optimizer hint with query operations.
- **FORCE_BULK=T** uses the SEM_APIS.BULK_LOAD_FROM_STAGING_TABLE procedure for bulk insertion of triples. This option may provide better performance on large updates.
- **LOAD_CLOB_ONLY=T** loads only triples/quads with object values longer than 4000 bytes in length when executing LOAD operations on N-Triple or N-Quad documents.
- **LOAD_OPTIONS={ OPTIONS_STRING }** uses `OPTIONS_STRING` as the extra file names when performing a LOAD operation.
- **MM_OPTIONS={ OPTIONS_STRING }** uses OPTIONS_STRING as the options parameter for operations calling SEM_APIS.MERGE_MODELS.
- **PARALLEL(n)** uses the SQL `PARALLEL(n)` hint for query and DML operations.

- **RESUME_LOAD=T** allows resuming an interrupted LOAD operation.
- **SERIALIZABLE=T** uses the SERIALIZABLE transaction isolation level for SEM_APIS.UPDATE_MODEL operations. READ COMMITTED is the default transaction isolation level.
- **STREAMING=F** materializes intermediate data and uses INSERT AS SELECT operations instead of streaming through JDBC Result Sets. This mode may provide better performance on large updates or updates with complex patterns in the WHERE clause.
- **STRICT_BNODE=F** enables ID-only operations for ADD, COPY, and MOVE. (ID-only operations are explained in Blank Nodes: Special Considerations for SPARQL Update.)

You can override some options settings at the session level by using the MDSYS.SDO_SEM_UPDATE_CTX.SET_PARAM procedure, as explained in Setting UPDATE_RDF_GRAPH Options at the Session Level.

For information about semantic network types and options, see RDF Networks.

Examples

The following example inserts six triples into a semantic model.

```
BEGIN
  sem_apis.update_model('electronics',
   'PREFIX : <http://www.example.org/electronics/>
    INSERT DATA {
       :camera1 :name "Camera 1" .
       :camera1 :price 120 .
       :camera1 :cameraType :Camera .
       :camera2 :name "Camera 2" .
       :camera2 :price 150 .
       :camera2 :cameraType :Camera .
      } ');
END;
/
```

15.157 SEM_APIS.UPDATE_RDF_GRAPH

Format

```
SEM_APIS.UPDATE_RDF_GRAPH(
     apply_rdf_graph      IN VARCHAR2,
     update_stmt          IN CLOB,
     match_models         IN SEM_MODELS DEFAULT NULL,
     match_rulebases      IN SEM_RULEBASES DEFAULT NULL,
     match_index_status   IN VARCHAR2 DEFAULT NULL,
     match_options        IN VARCHAR2 DEFAULT NULL,
     options              IN VARCHAR2 DEFAULT NULL,
     network_owner        IN VARCHAR2 DEFAULT NULL,
     network_name         IN VARCHAR2 DEFAULT NULL);
```

Description

Executes a SPARQL update statement on an RDF graph.

Parameters

apply_rdf_graph
Name of the RDF graph to be updated. This is the name specified when the graph was created using the SEM_APIS.CREATE_RDF_GRAPH procedure.
It cannot be an RDF graph collection (see RDF Graph Collections) or an RDF view.

update_stmt
One or more SPARQL update commands to be executed on the `apply_rdf_graph` graph. Use the semicolon (;) to separate commands.

match_models
A list of RDF graphs that forms the SPARQL data set to query for graph pattern matching during a SPARQL update operation (INSERT WHERE, DELETE WHERE, COPY, MOVE, ADD). Can include RDF graph collections and/or RDF views If this parameter is not specified, the `apply_rdf_graph` graph is used.

match_rulebases
A list of rulebases to use with `match_models` to provide an inferred graph that generates additional triples or quads to use for graph pattern matching during a SPARQL update operation.

match_index_status
The desired status for any inferred graphs used for graph pattern matching during a SPARQL update operation.

match_options
String specifying hints to influence graph pattern matching during a SPARQL update operation. The set of hints that can be used here is identical to those that can be used in the `options` parameter of SEM_MATCH.

options
String specifying hints that affect SPARQL operations. See the Usage Notes for a list of available options.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

Before using this procedure, be sure you understand the material in Support for SPARQL Update Operations on an RDF Graph.

The `options` parameter can specify one or more of the following options:

- **APP_TAB_IDX={INDEX_NAME}** uses an INDEX optimizer hint for INDEX_NAME when doing DML operations on the application table.
- **APPEND** uses the SQL APPEND hint with DML operations.
- **AUTOCOMMIT=F** avoids starting and committing a transaction for each SEM_APIS.UPDATE_RDF_GRAPH call. Instead, this option gives transaction control to the caller. Each SEM_APIS.UPDATE_RDF_GRAPH call will execute as part of a main transaction that is started, committed, or rolled back by the caller.

- **BULK_OPTIONS={OPTIONS_STRING}** uses OPTIONS_STRING as the `flags` parameter when calling SEM_APIS.BULK_LOAD_FROM_STAGING_TABLE.
- **CLOB_UPDATE_SUPPORT=T** turns on CLOB functionality.
- **DEL_AS_INS=T** performs a large delete operation by inserting all data that should remain after the delete operation instead of doing deletions. This option may significantly improve the performance of large delete operations.
- **DYNAMIC_SAMPLING(n)** uses DYNAMIC_SAMPLING(n) SQL optimizer hint with query operations.
- **FORCE_BULK=T** uses the SEM_APIS.BULK_LOAD_RDF_GRAPH procedure for bulk insertion of triples. This option may provide better performance on large updates.
- **LOAD_CLOB_ONLY=T** loads only triples/quads with object values longer than 4000 bytes in length when executing LOAD operations on N-Triple or N-Quad documents.
- **LOAD_OPTIONS={ OPTIONS_STRING }** uses `OPTIONS_STRING` as the extra file names when performing a LOAD operation.
- **MM_OPTIONS={ OPTIONS_STRING }** uses OPTIONS_STRING as the options parameter for operations calling SEM_APIS.MERGE_RDF_GRAPHS.
- **PARALLEL(n)** uses the SQL `PARALLEL(n)` hint for query and DML operations.
- **RESUME_LOAD=T** allows resuming an interrupted LOAD operation.
- **SERIALIZABLE=T** uses the SERIALIZABLE transaction isolation level for SEM_APIS.UPDATE_RDF_GRAPH operations. READ COMMITTED is the default transaction isolation level.
- **STREAMING=F** materializes intermediate data and uses INSERT AS SELECT operations instead of streaming through JDBC Result Sets. This mode may provide better performance on large updates or updates with complex patterns in the WHERE clause.
- **STRICT_BNODE=F** enables ID-only operations for ADD, COPY, and MOVE. (ID-only operations are explained in Blank Nodes: Special Considerations for SPARQL Update.)

You can override some options settings at the session level by using the MDSYS.SDO_SEM_UPDATE_CTX.SET_PARAM procedure, as explained in Setting UPDATE_RDF_GRAPH Options at the Session Level.

For information about RDF network types and options, see RDF Networks.

Examples

The following example inserts six triples into an RDF graph.

```
BEGIN
  sem_apis.update_rdf_graph('electronics',
   'PREFIX : <http://www.example.org/electronics/>
    INSERT DATA {
       :camera1 :name "Camera 1" .
       :camera1 :price 120 .
       :camera1 :cameraType :Camera .
       :camera2 :name "Camera 2" .
       :camera2 :price 150 .
       :camera2 :cameraType :Camera .
      } ');
END;
/
```

15.158 SEM_APIS.VALIDATE_ENTAILMENT

Format

```
SEM_APIS.VALIDATE_ENTAILMENT(
     models_in       IN SEM_MODELS,
     rulebases_in    IN SEM_RULEBASES,
     criteria_in     IN VARCHAR2 DEFAULT NULL,
     max_conflict    IN NUMBER DEFAULT 100,
     options         IN VARCHAR2 DEFAULT NULL,
     network_owner   IN VARCHAR2 DEFAULT NULL,
     network_name    IN VARCHAR2 DEFAULT NULL
     ) RETURN RDF_LONGVARCHARARRAY;
```

> **Note:**
>
> This subprogram will be deprecated in a future release. It is recommended that you use the SEM_APIS.VALIDATE_INFERRED_GRAPH subprogram instead.

Description

Validates entailments (rules indexes) that can be used to perform OWL or RDFS inferencing for one or more models.

Parameters

models_in
One or more model names. Its data type is SEM_MODELS, which has the following definition: `TABLE OF VARCHAR2(25)`

rulebases_in
One or more rulebase names. Its data type is SEM_RULEBASES, which has the following definition: `TABLE OF VARCHAR2(25)`. Rules and rulebases are explained in Inferencing: Rules and Rulebases.

criteria_in
A comma-delimited string of validation checks to run. If you do not specify this parameter, by default all of the following checks are run:

- `UNSAT`: Find unsatisfiable classes.
- `EMPTY`: Find instances that belong to unsatisfiable classes.
- `SYNTAX_S`: Find triples whose subject is neither URI nor blank node.
- `SYNTAX_P`: Find triples whose predicate is not URI.
- `SELF_DIF`: Find individuals that are different from themselves.
- `INST`: Find individuals that simultaneously belong to two disjoint classes.
- `SAM_DIF`: Find pairs of individuals that are same (owl:sameAs) and different (owl:differentFrom) at the same time.

To specify fewer checks, specify a string with only the checks to be performed. For example, `criteria_in => 'UNSAT'` causes the validation process to search only for unsatisfiable classes.

max_conflict
The maximum number of conflicts to find before the validation process stops. The default value is 100.

options
(Not currently used. Reserved for Oracle use.).

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

This procedure can be used to detect inconsistencies in the original entailment. For more information, see Validating OWL Models and Entailments.

This procedure returns a null value if no errors are detected or (if errors are detected) an object of type RDF_LONGVARCHARARRAY, which has the following definition: `VARRAY(32767) OF VARCHAR2(4000)`

To create an entailment, use the SEM_APIS.CREATE_ENTAILMENT procedure.

For information about semantic network types and options, see RDF Networks.

Examples

For an example of this procedure, see Example 3-5 in Validating OWL Models and Entailments.

15.159 SEM_APIS.VALIDATE_GEOMETRIES

Format

```
SEM_APIS.VALIDATE_GEOMETRIES(
     model_name      IN VARCHAR2,
     SRID            IN NUMBER,
     tolerance       IN NUMBER,
     parallel        IN PLS_INTEGER DEFAULT NULL,
     tablespace_name IN VARCHAR2 DEFAULT NULL,
     options         IN VARCHAR2 DEFAULT NULL,
     network_owner  IN VARCHAR2 DEFAULT NULL,
     network_name   IN VARCHAR2 DEFAULT NULL);
```

Description

Determines if all geometry literals in the specified model are valid for the provided SRID and tolerance values.

Parameters

model_name
Name of the model containing geometry literals to validate. Only native models can be specified.

SRID
SRID for the spatial reference system.

tolerance
Tolerance value that should be used for validation.

parallel
Degree of parallelism to be associated with the operation. For more information about parallel execution, see *Oracle Database VLDB and Partitioning Guide*.

tablespace_name
Destination tablespace for the tables *{model_name}*_IVG$, *{model_name}*_FXT$, and *{model_name}*_NFT$.

options
String specifying options for validation. Supported options are:

- RECTIFY=T. Staging tables *{model_name}*_FXT$ and *{model_name}*_NFT$ are created, containing rectifiable and non-rectifiable triples, respectively. You can use these tables to correct the model.
- AUTOCORRECT=T. Triples containing invalid but rectifiable geometries are corrected. Also, table *{model_name}*_NFT$ containing triples with non-rectifiable geometries is created so that you can correct such triples manually.
- STANDARD_CRS_URI=T. Use standard CRS (coordinate reference systems) URIs.
- GML_LIT_SRL=T. Use `ogc:gmlLiteral` serialization for corrected geometry literals. `ogc:wktLiteral` serialization is the default.
- GEOJSON_LIT_SRL=T. Use `ogc:geoJSONLiteral` serialization for corrected geometry literals. `ogc:wktLiteral` serialization is the default.
- KML_LIT_SRL=T. Use `ogc:kmlLiteral` serialization for corrected geometry literals. `ogc:wktLiteral` serialization is the default.

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

This procedure is a wrapper for SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT function.

A table *{model_name}*_IVG$ containing invalid geometry literals is created. Optionally, staging tables *{model_name}*_FXT$ and *{model_name}*_NFT$ can be created, containing rectifiable and non-rectifiable triples, respectively. Staging tables allow the user to correct invalid geometries. Invalid but rectifiable geometry literals in a model can also be rectified automatically if specified.

After correction of invalid geometries in a model, it is recommended that you execute SEM_APIS.PURGE_UNUSED_VALUES to purge invalid geometry literal values from the semantic network.

For an explanation of models, see Semantic Data Modeling and Semantic Data in the Database.

For information about semantic network types and options, see RDF Networks.

Examples

The following example creates a model with some invalid geometry literals and then validates the model using the `RECTIFY=T` and `STANDARD_CRS_URI=T` options.

```
-- Create model
EXEC sem_apis.create_sem_model('m', NULL, NULL, network_owner=>'RDFUSER',
network_name=>'NET1');

-- Insert invalid geometries
-- Duplicated coordinates - rectifiable
insert into RDFUSER.NET1#RDFT_M(triple) values (sdo_rdf_triple_s('m','<http://my.org/
geom1>', '<http://www.opengis.net/rdf#asWKT>', '"POLYGON((1.0 2.0, 3.0 2.0, 1.0 4.0, 1.0
2.0, 1.0 2.0))"^^<http://xmlns.oracle.com/rdf/geo/WKTLiteral>'),
network_owner=>'RDFUSER', network_name=>'NET1');
-- Boundary is not closed - rectifiable
insert into RDFUSER.NET1#RDFT_M(triple) values (sdo_rdf_triple_s('m','<http://my.org/
geom2>', '<http://www.opengis.net/rdf#asWKT>', '"POLYGON((1.0 2.0, 3.0 2.0, 3.0 4.0, 1.0
4.0))"^^<http://xmlns.oracle.com/rdf/geo/WKTLiteral>'), network_owner=>'RDFUSER',
network_name=>'NET1');
-- Less than 4 points - non rectifiable
insert into RDFUSER.NET1#RDFT_M(triple) values (sdo_rdf_triple_s('m:<http://my.org/
g2>','<http://my.org/geom3>', '<http://www.opengis.net/rdf#asWKT>', '"POLYGON((1.0 2.0,
3.0 2.0, 1.0 4.0))"^^<http://xmlns.oracle.com/rdf/geo/WKTLiteral>'),
network_owner=>'RDFUSER', network_name=>'NET1');
commit;

-- Validate
EXEC sem_apis.validate_geometries(model_name=>'m',SRID=>8307,tolerance=>0.1,
options=>'STANDARD_CRS_URI=T RECTIFY=T', network_owner=>'RDFUSER', network_name=>'NET1');

-- Check invalid geometries
SELECT original_vid, error_msg, corrected_geom_literal FROM M_IVG$;

-- Check rectified triples
select RDF$STC_GRAPH, RDF$STC_SUB, RDF$STC_PRED, RDF$STC_OBJ from M_FXT$;

-- Check non-rectified triples
select RDF$STC_GRAPH, RDF$STC_SUB, RDF$STC_PRED, RDF$STC_OBJ, ERROR_MSG from M_NFT$;
```

15.160 SEM_APIS.VALIDATE_INFERRED_GRAPH

Format

```
SEM_APIS.VALIDATE_INFERRED_GRAPH(
     rdf_graphs_in   IN SEM_MODELS,
     rulebases_in    IN SEM_RULEBASES,
     criteria_in     IN VARCHAR2 DEFAULT NULL,
     max_conflict    IN NUMBER DEFAULT 100,
     options         IN VARCHAR2 DEFAULT NULL,
     network_owner   IN VARCHAR2 DEFAULT NULL,
     network_name    IN VARCHAR2 DEFAULT NULL
     ) RETURN RDF_LONGVARCHARARRAY;
```

Description

Validates inferred graphs (rules indexes) that can be used to perform OWL or RDFS inferencing for one or more RDF graphs.

Parameters

rdf_graphs_in
One or more RDF graph names. Its data type is SEM_MODELS, which has the following definition: `TABLE OF VARCHAR2(25)`

rulebases_in
One or more rulebase names. Its data type is SEM_RULEBASES, which has the following definition: `TABLE OF VARCHAR2(25)`. Rules and rulebases are explained in Inferencing: Rules and Rulebases.

criteria_in
A comma-delimited string of validation checks to run. If you do not specify this parameter, by default all of the following checks are run:

- `UNSAT`: Find unsatisfiable classes.
- `EMPTY`: Find instances that belong to unsatisfiable classes.
- `SYNTAX_S`: Find triples whose subject is neither URI nor blank node.
- `SYNTAX_P`: Find triples whose predicate is not URI.
- `SELF_DIF`: Find individuals that are different from themselves.
- `INST`: Find individuals that simultaneously belong to two disjoint classes.
- `SAM_DIF`: Find pairs of individuals that are same (owl:sameAs) and different (owl:differentFrom) at the same time.

To specify fewer checks, specify a string with only the checks to be performed. For example, `criteria_in => 'UNSAT'` causes the validation process to search only for unsatisfiable classes.

max_conflict
The maximum number of conflicts to find before the validation process stops. The default value is 100.

options
(Not currently used. Reserved for Oracle use.).

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

This procedure can be used to detect inconsistencies in the original inferred graph. For more information, see Validating OWL Models and Entailments.

This procedure returns a null value if no errors are detected or (if errors are detected) an object of type RDF_LONGVARCHARARRAY, which has the following definition: `VARRAY(32767) OF VARCHAR2(4000)`

To create an inferred graph, use the SEM_APIS.CREATE_INFERRED_GRAPH procedure.

For information about RDF network types and options, see RDF Networks.

Examples

For an example of this procedure, see Example 3-5 in Validating OWL Models and Entailments.

15.161 SEM_APIS.VALIDATE_MODEL

Format

```
SEM_APIS.VALIDATE_MODEL(
     models_in    IN SEM_MODELS,
     criteria_in  IN VARCHAR2 DEFAULT NULL,
     max_conflict IN NUMBER DEFAULT 100,
     options      IN VARCHAR2 DEFAULT NULL
     ) RETURN RDF_LONGVARCHARARRAY;
```

> **Note:**
>
> This subprogram will be deprecated in a future release. It is recommended that you use the SEM_APIS.VALIDATE_RDF_GRAPH subprogram instead.

Description

Validates one or more models.

Parameters

models_in
One or more model names. Its data type is SEM_MODELS, which has the following definition: `TABLE OF VARCHAR2(25)`

criteria_in
A comma-delimited string of validation checks to run. If you do not specify this parameter, by default all of the following checks are run:

- `UNSAT`: Find unsatisfiable classes.
- `EMPTY`: Find instances that belong to unsatisfiable classes.
- `SYNTAX_S`: Find triples whose subject is neither URI nor blank node.
- `SYNTAX_P`: Find triples whose predicate is not URI.
- `SELF_DIF`: Find individuals that are different from themselves.
- `INST`: Find individuals that simultaneously belong to two disjoint classes.
- `SAM_DIF`: Find pairs of individuals that are same (owl:sameAs) and different (owl:differentFrom) at the same time.

To specify fewer checks, specify a string with only the checks to be performed. For example, `criteria_in => 'UNSAT'` causes the validation process to search only for unsatisfiable classes.

max_conflict
The maximum number of conflicts to find before the validation process stops. The default value is 100.

options
(Not currently used. Reserved for Oracle use.).

Usage Notes

This procedure can be used to detect inconsistencies in the original data model. For more information, see Validating OWL Models and Entailments.

This procedure returns a null value if no errors are detected or (if errors are detected) an object of type RDF_LONGVARCHARARRAY, which has the following definition: `VARRAY(32767) OF VARCHAR2(4000)`

Examples

The following example validates the model named `family`.

```
SELECT SEM_APIS.VALIDATE_MODEL(SEM_MODELS('family')) FROM DUAL;
```

15.162 SEM_APIS.VALIDATE_RDF_GRAPH

Format

```
SEM_APIS.VALIDATE_RDF_GRAPH(
     rdf_graphs_in     IN SEM_MODELS,
     criteria_in       IN VARCHAR2 DEFAULT NULL,
     max_conflict      IN NUMBER DEFAULT 100,
     options           IN VARCHAR2 DEFAULT NULL,
     network_owner     IN VARCHAR2(128) DEFAULT,
     network_name      IN VARCHAR2 DEFAULT
    ) RETURN RDF_LONGVARCHARARRAY;
```

Description

Validates one or more RDF graphs.

Parameters

rdf_graphs_in
One or more RDF graph names. Its data type is SEM_MODELS, which has the following definition: `TABLE OF VARCHAR2(25)`

criteria_in
A comma-delimited string of validation checks to run. If you do not specify this parameter, by default all of the following checks are run:

- `UNSAT`: Find unsatisfiable classes.
- `EMPTY`: Find instances that belong to unsatisfiable classes.
- `SYNTAX_S`: Find triples whose subject is neither URI nor blank node.
- `SYNTAX_P`: Find triples whose predicate is not URI.
- `SELF_DIF`: Find individuals that are different from themselves.
- `INST`: Find individuals that simultaneously belong to two disjoint classes.
- `SAM_DIF`: Find pairs of individuals that are same (owl:sameAs) and different (owl:differentFrom) at the same time.

To specify fewer checks, specify a string with only the checks to be performed. For example, `criteria_in => 'UNSAT'` causes the validation process to search only for unsatisfiable classes.

max_conflict
The maximum number of conflicts to find before the validation process stops. The default value is 100.

options
(Not currently used. Reserved for Oracle use.).

network_owner
Owner of the RDF network. (See Table 1-2.)

network_name
Name of the RDF network. (See Table 1-2.)

Usage Notes

This procedure can be used to detect inconsistencies in the original RDF graph. For more information, see Validating OWL Models and Entailments.

This procedure returns a null value if no errors are detected or (if errors are detected) an object of type RDF_LONGVARCHARARRAY, which has the following definition: `VARRAY(32767) OF VARCHAR2(4000)`

Examples

The following example validates the RDF graph named `family`.

```
SELECT SEM_APIS.VALIDATE_RDF_GRAPH(SEM_MODELS('family')) FROM DUAL;
```

15.163 SEM_APIS.VALUE_NAME_PREFIX

Format

```
SEM_APIS.VALUE_NAME_PREFIX (
     value_name  IN VARCHAR2,
     value_type  IN VARCHAR2
     ) RETURN VARCHAR2;
```

Description

Returns the value in the VNAME_PREFIX column for the specified value name and value type pair in the RDF_VALUE$ table.

Parameters

value_name
Value name. Must match a value in the VALUE_NAME column in the RDF_VALUE$ table, which is described in Statements.

value_type
Value type. Must match a value in the VALUE_TYPE column in the RDF_VALUE$ table, which is described in Statements.

Usage Notes

This function usually causes an index on the RDF_VALUE$ table to be used for processing a lookup for values, and thus can make a query run faster.

Examples

The following query returns value name portions of all the lexical values in RDF_VALUE$ table with a prefix value same as that returned by the VALUE_NAME_PREFIX function. This query uses an index on the RDF_VALUE$ table, thereby providing efficient lookup.

```
SELECT value_name FROM RDF_VALUE$
  WHERE vname_prefix = SEM_APIS.VALUE_NAME_PREFIX(
    'http://www.w3.org/1999/02/22-rdf-syntax-ns#type','UR');

VALUE_NAME
--------------------------------------------------------------------------------
http://www.w3.org/1999/02/22-rdf-syntax-ns#Alt
http://www.w3.org/1999/02/22-rdf-syntax-ns#Bag
http://www.w3.org/1999/02/22-rdf-syntax-ns#List
http://www.w3.org/1999/02/22-rdf-syntax-ns#Property
http://www.w3.org/1999/02/22-rdf-syntax-ns#Seq
http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement
http://www.w3.org/1999/02/22-rdf-syntax-ns#XMLLiteral
http://www.w3.org/1999/02/22-rdf-syntax-ns#first
http://www.w3.org/1999/02/22-rdf-syntax-ns#nil
http://www.w3.org/1999/02/22-rdf-syntax-ns#object
http://www.w3.org/1999/02/22-rdf-syntax-ns#predicate
http://www.w3.org/1999/02/22-rdf-syntax-ns#rest
http://www.w3.org/1999/02/22-rdf-syntax-ns#subject
http://www.w3.org/1999/02/22-rdf-syntax-ns#type
http://www.w3.org/1999/02/22-rdf-syntax-ns#value

15 rows selected.
```

15.164 SEM_APIS.VALUE_NAME_SUFFIX

Format

```
SEM_APIS.VALUE_NAME_SUFFIX (
     value_name   IN VARCHAR2,
     value_type   IN VARCHAR2
     ) RETURN VARCHAR2;
```

Description

Returns the value in the VNAME_SUFFIX column for the specified value name and value type pair in the RDF_VALUE$ table.

Parameters

value_name
Value name. Must match a value in the VALUE_NAME column in the RDF_VALUE$ table, which is described in Statements.

value_type
Value type. Must match a value in the VALUE_TYPE column in the RDF_VALUE$ table, which is described in Statements.

Usage Notes

This function usually causes an index on the RDF_VALUE$ table to be used for processing a lookup for values, and thus can make a query run faster.

Examples

The following query returns value name portions of all the lexical values in RDF_VALUE$ table with a suffix value same as that returned by the VALUE_NAME_SUFFIX function. This query uses an index on the RDF_VALUE$ table, thereby providing efficient lookup.

```
SELECT value_name FROM RDF_VALUE$
  WHERE vname_suffix = SEM_APIS.VALUE_NAME_SUFFIX(
    'http://www.w3.org/1999/02/22-rdf-syntax-ns#type','UR');

VALUE_NAME
--------------------------------------------------------------------------------
http://www.w3.org/1999/02/22-rdf-syntax-ns#type
```

16 SEM_PERF Package Subprograms

The SEM_PERF package contains subprograms for examining and enhancing the performance of the Resource Description Framework (RDF) and Web Ontology Language (OWL) support in an Oracle database.

To use the subprograms in this chapter, you must understand the conceptual and usage information in RDF Semantic Graph Overview and OWL Concepts.

This chapter provides reference information about the subprograms, listed in alphabetical order.

- SEM_PERF.ANALYZE_AUX_TABLES
- SEM_PERF.DELETE_NETWORK_STATS
- SEM_PERF.DROP_EXTENDED_STATS
- SEM_PERF.EXPORT_NETWORK_STATS
- SEM_PERF.GATHER_STATS
- SEM_PERF.IMPORT_NETWORK_STATS

16.1 SEM_PERF.ANALYZE_AUX_TABLES

Format

```
SEM_PERF.ANALYZE_AUX_TABLES(
     model_name       IN VARCHAR2,
     estimate_percent IN NUMBER DEFAULT DBMS_STATS.AUTO_SAMPLE_SIZE,
     method_opt       IN VARCHAR2 DEFAULT NULL,
     degree           IN NUMBER DEFAULT DBMS_STATS.AUTO_DEGREE,
     network_owner    IN DBMS_ID  DEFAULT NULL,
     network_name     IN VARCHAR2 DEFAULT NULL);
```

Description

Analyzes all the SPM tables currently present for the given RDF model.

Parameters

model_name
Name of the RDF model.

estimate_percent
Determines the percentage of rows to sample. For more information on gathering the `estimate_percent` statistics, see DBMS_STATS.GATHER_TABLE_STATS procedure.

method_opt
Determines the column statistics collection. For more information on gathering the column statistics, see DBMS_STATS.GATHER_TABLE_STATS

degree
Determines the degree of parallelism used for gathering statistics. For more information on this procedure parameter see DBMS_STATS.GATHER_TABLE_STATS

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

Examples

The following example gathers statistics for SPM auxiliary tables.

```
EXECUTE SEM_PERF.ANALYZE_AUX_TABLES('m1',network_owner=>'RDFUSER',network_name=>'NET1');
```

16.2 SEM_PERF.DELETE_NETWORK_STATS

Format

```
SEM_PERF.DELETE_NETWORK_STATS (
     cascade_parts     IN BOOLEAN DEFAULT TRUE,
     cascade_columns   IN BOOLEAN DEFAULT TRUE,
     cascade_indexes   IN BOOLEAN DEFAULT TRUE,
     no_invalidate     IN BOOLEAN DEFAULT DBMS_STATS.AUTO_INVALIDATE,
     force             IN BOOLEAN DEFAULT FALSE,
     options           IN VARCHAR2 DEFAULT NULL,
     network_owner     IN VARCHAR2 DEFAULT NULL,
     network_name      IN VARCHAR2 DEFAULT NULL);
```

Description

Deletes statistics for the semantic network.

Parameters

options
Controls the scope of the operation:

- If `MDSYS.SDO_RDF.VALUE_TABLE_ONLY`, the operation applies only to the RDF_VALUE$ table.
- If `MDSYS.SDO_RDF.LINK_TABLE_ONLY`, the operation applies only to the RDF_LINK$ table.
- If null (the default), the operation applies to both the RDF_VALUE$ and RDF_LINK$ tables.

(other parameters)
See the parameter explanations for the DBMS_STATS.DELETE_TABLE_STATS procedure in *Oracle Database PL/SQL Packages and Types Reference*, although `force` here applies to network statistics.

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

See the information about the DBMS_STATS package in*Oracle Database PL/SQL Packages and Types Reference*.

See also Managing Statistics for Semantic Models and the Semantic Network.

For information about semantic network types and options, see RDF Networks.

Examples

The following example deletes statistics for the semantic network:

```
EXECUTE SEM_APIS.DELETE_NETWORK_STATS;
```

16.3 SEM_PERF.DROP_EXTENDED_STATS

Format

```
SEM_PERF.DROP_EXTENDED_STATS (
     network_owner IN VARCHAR2 DEFAULT NULL,
     network_name  IN VARCHAR2 DEFAULT NULL);
```

Description

Drops column groups used for extended optimizer statistics on the RDF_LINK$ table.

Parameters

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

To use this procedure, you must connect as a user with permission to execute it. Network owners and DBAs have privileges to execute this procedure.

The default column groups that will be dropped from RDF_LINK$ are: `(CANON_END_NODE_ID, START_NODE_ID) (P_VALUE_ID, CANON_END_NODE_ID) (P_VALUE_ID, START_NODE_ID)`

See also:

- Dropping Extended Statistics at the Network Level
- The information about the DBMS_STATS package in *Oracle Database PL/SQL Packages and Types Reference*

For information about semantic network types and options, see RDF Networks.

Examples

The following example drops extended statistics for the semantic network named NET1 owned by RDFUSER:

```
EXECUTE SEM_PERF.DROP_EXTENDED_STATS(network_owner=>'RDFUSER', network_name=>'NET1');
```

16.4 SEM_PERF.EXPORT_NETWORK_STATS

Format

```
SEM_PERF.EXPORT_NETWORK_STATS (
     stattab        IN VARCHAR2,
     statid         IN VARCHAR2 DEFAULT NULL,
     cascade        IN BOOLEAN DEFAULT TRUE,
     statown        IN VARCHAR2 DEFAULT NULL,
     stat_category  IN VARCHAR2 DEFAULT 'OBJECT_STATS',
     options        IN VARCHAR2 DEFAULT NULL,
     network_owner  IN VARCHAR2 DEFAULT NULL,
     network_name   IN VARCHAR2 DEFAULT NULL);
```

Description

Exports the statistics for the semantic network and stores them in the user statistics table.

Parameters

options
Controls the scope of the operation:

- If `MDSYS.SDO_RDF.VALUE_TABLE_ONLY`, the operation applies only to the RDF_VALUE$ table.
- If `MDSYS.SDO_RDF.LINK_TABLE_ONLY`, the operation applies only to the RDF_LINK$ table.
- If null (the default), the operation applies to both the RDF_VALUE$ and RDF_LINK$ tables.

(other parameters)
See the parameter explanations for the DBMS_STATS.EXPORT_TABLE_STATS procedure in *Oracle Database PL/SQL Packages and Types Reference*.

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

See the information about the DBMS_STATS package in*Oracle Database PL/SQL Packages and Types Reference*.

See also Managing Statistics for Semantic Models and the Semantic Network.

For information about semantic network types and options, see RDF Networks.

Examples

The following example exports the statistics for the semantic network and stores them in a table named `STAT_TABLE`.

```
EXECUTE SEM_APIS.EXPORT_NETWORK_STATS('stat_table', network_owner=>'RDFUSER',
network_name=>'NET1');
```

16.5 SEM_PERF.GATHER_STATS

Format

```
SEM_PERF.GATHER_STATS(
     just_on_values_table  IN BOOLEAN DEFAULT FALSE,
     degree                IN NUMBER(38) DEFAULT NULL,
     estimate_percent      IN NUMBER DEFAULT DBMS_STATS.AUTO_SAMPLE_SIZE,
     value_method_opt      IN VARCHAR2 DEFAULT NULL,
     link_method_opt       IN VARCHAR2 DEFAULT NULL,
     network_owner         IN VARCHAR2 DEFAULT NULL,
     network_name          IN VARCHAR2 DEFAULT NULL);
```

Description

Gathers statistics about RDF and OWL tables and their indexes.

Parameters

just_on_values_table
`TRUE` collects statistics only on the table containing the lexical values of triples; `FALSE` (the default) collects statistics on all major tables related to the storage of RDF and OWL data.
A value of `TRUE` reduces the execution time for the procedure; and it may be sufficient if you need only to collect statistics on the values table (for example, if you use other interfaces to collect any other statistics that you might need).

degree
Degree of parallelism. For more information about parallel execution, see *Oracle Database VLDB and Partitioning Guide*.

estimate_percent
Determines the percentage of rows in RDF_LINK$ and RDF_VALUE$ to sample.
The valid range is between 0.000001 and 100. You can use the constant `DBMS_STATS.AUTO_SAMPLE_SIZE` (the default) to enable Oracle Database to determine the appropriate sample size for optimal statistics.

value_method_opt
Accepts either of the following options, or both in combination, for the RDF_VALUE$ table:

- `FOR ALL [INDEXED | HIDDEN] COLUMNS [size_clause]`
- `FOR COLUMNS [size clause] column|attribute [size_clause] [,column|attribute [size_clause]...]`

`size_clause` is defined as: `size_clause := SIZE {integer | REPEAT | AUTO | SKEWONLY}`
`column` is defined as: `column := column_name | (extension)`

- `integer` : Number of histogram buckets. Must be in the range [1, 2048].
- `REPEAT` : Collects histograms only on the columns that already have histograms.
- `AUTO` : Oracle Database determines the columns to collect histograms based on data distribution and the workload of the columns.
- `SKEWONLY` : Oracle Database determines the columns to collect histograms based on the data distribution of the columns.
- `column_name` : name of a column

- `extension`: Can be either a column group in the format of (column_name, column_name [, ...]) or an expression.

The usual default is: `FOR ALL COLUMNS SIZE 2048`

link_method_opt
Accepts either of the following options, or both in combination, for the RDF_LINK$ table:

- `FOR ALL [INDEXED | HIDDEN] COLUMNS [size_clause]`
- `FOR COLUMNS [size clause] column|attribute [size_clause] [,column|attribute [size_clause]...]`

`size_clause` is defined as: `size_clause := SIZE {integer | REPEAT | AUTO | SKEWONLY}`
`column` is defined as: `column := column_name | (extension)`

- `integer` : Number of histogram buckets. Must be in the range [1,2048].
- `REPEAT` : Collects histograms only on the columns that already have histograms.
- `AUTO` : Oracle Database determines the columns to collect histograms based on data distribution and the workload of the columns.
- `SKEWONLY` : Oracle Database determines the columns to collect histograms based on the data distribution of the columns.
- `column_name` : Name of a column.
- `extension`: Can be either a column group in the format of `(column_name, column_name [, ...])` or an expression.

The usual default is: `FOR ALL COLUMNS SIZE AUTO FOR COLUMNS SIZE 2048 P_VALUE_ID CANON_END_NODE_ID START_NODE_ID G_ID (CANON_END_NODE_ID, START_NODE_ID) (P_VALUE_ID, CANON_END_NODE_ID) (P_VALUE_ID, START_NODE_ID)`

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

To use this procedure, you must connect as a user with permission to execute it. Network owners and DBAs have privileges execute this procedure.

This procedure collects statistical information that can help you to improve inferencing performance, as explained in Enhancing Inference Performance. This procedure internally calls the DBMS_STATS.GATHER_TABLE_STATS procedure to collect statistics on RDF- and OWL-related tables and their indexes, and stores the statistics in the Oracle Database data dictionary. For information about using the DBMS_STATS package, see *Oracle Database PL/SQL Packages and Types Reference*.

Gathering statistics uses significant system resources, so execute this procedure when it cannot adversely affect essential applications and operations.

See also Managing Statistics for Semantic Models and the Semantic Network.

Examples

The following example gathers statistics about RDF and OWL related tables and their indexes.

```
EXECUTE SEM_PERF.GATHER_STATS(network_owner=>'RDFUSER', network_name=>'NET1');
```

16.6 SEM_PERF.IMPORT_NETWORK_STATS

Format

```
SEM_PERF.IMPORT_NETWORK_STATS (
     stattab        IN VARCHAR2,
     statid         IN VARCHAR2 DEFAULT NULL,
     cascade        IN BOOLEAN DEFAULT TRUE,
     statown        IN VARCHAR2 DEFAULT NULL,
     no_invalidate  IN BOOLEAN DEFAULT FALSE,
     force          IN BOOLEAN DEFAULT FALSE,
     stat_category  IN VARCHAR2 DEFAULT 'OBJECT_STATS',
     options        IN VARCHAR2 DEFAULT NULL,
     network_owner  IN VARCHAR2 DEFAULT NULL,
     network_name   IN VARCHAR2 DEFAULT NULL);
```

Description

Retrieves the statistics for the semantic network from a user statistics table and stores them in the dictionary.

Parameters

options
Controls the scope of the operation:

- If `MDSYS.SDO_RDF.VALUE_TABLE_ONLY`, the operation applies only to the RDF_VALUE$ table.
- If `MDSYS.SDO_RDF.LINK_TABLE_ONLY`, the operation applies only to the RDF_LINK$ table.
- If null (the default), the operation applies to both the RDF_VALUE$ and RDF_LINK$ tables.

(other parameters)
See the parameter explanations for the DBMS_STATS.IMPORT_TABLE_STATS procedure in *Oracle Database PL/SQL Packages and Types Reference*, although `force` here applies to network statistics.

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

See the information about the DBMS_STATS package in*Oracle Database PL/SQL Packages and Types Reference*.

See also Managing Statistics for Semantic Models and the Semantic Network.

For information about semantic network types and options, see RDF Networks.

Examples

The following example imports the statistics for the semantic network in a table named `STAT_TABLE`, and stores them in the dictionary.

```
EXECUTE SEM_APIS.IMPORT_NETWORK_STATS('stat_table', network_owner=>'RDFUSER',
network_name=>'NET1');
```

17 SEM_RDFCTX Package Subprograms

The SEM_RDFCTX package contains subprograms (functions and procedures) to manage extractor policies and semantic indexes created for documents.

To use the subprograms in this chapter, you should understand the conceptual and usage information in Semantic Indexing for Documents.

This chapter provides reference information about the subprograms, listed in alphabetical order.

- SEM_RDFCTX.ADD_DEPENDENT_POLICY
- SEM_RDFCTX.CREATE_POLICY
- SEM_RDFCTX.DROP_POLICY
- SEM_RDFCTX.MAINTAIN_TRIPLES
- SEM_RDFCTX.SET_DEFAULT_POLICY
- SEM_RDFCTX.SET_EXTRACTOR_PARAM

17.1 SEM_RDFCTX.ADD_DEPENDENT_POLICY

Format

```
SEM_RDFCTX.ADD_DEPENDENT_POLICY(
     index_name     IN VARCHAR2,
     policy_name    IN VARCHAR2,
     partition_name IN VARCHAR2 DEFAULT NULL,
     network_owner  IN VARCHAR2 DEFAULT NULL,
     network_name   IN VARCHAR2 DEFAULT NULL);
```

Description

Adds a dependent policy to an (already created) index or index partition.

Parameters

index_name
Name of the index.

policy_name
Name of the dependent policy.

partition_name
If the specified index is local, the name of the target partition. (Otherwise, must be null.)

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

The base policy corresponding to the new dependent policy must already be a part of the index.

For information about semantic network types and options, see RDF Networks.

Examples

The following example adds a new dependent policy `SEM_EXTR_PLUS_GEOONT` to the index `ArticleIndex`.

```
begin
  sem_rdfctx.add_dependent_policy (index_name  => 'ArticleIndex',
                                   policy_name => 'SEM_EXTR_PLUS_GEOONT');
end;
/
```

17.2 SEM_RDFCTX.CREATE_POLICY

Format

```
SEM_RDFCTX.CREATE_POLICY(
     policy_name   IN VARCHAR2,
     extractor     IN mdsys.rdfctx_extractor,
     preferences   IN sys.XMLType DEFAULT NULL,
     network_owner IN VARCHAR2 DEFAULT NULL,
     network_name  IN VARCHAR2 DEFAULT NULL);
```

or

```
SEM_RDFCTX.CREATE_POLICY(
     policy_name       IN VARCHAR2,
     base_policy       IN VARCHAR2,
     user_models       IN SEM_MODELS DEFAULT NULL,
     user_entailments  IN SEM_MODELS DEFAULT NULL,
     network_owner     IN VARCHAR2 DEFAULT NULL,
     network_name      IN VARCHAR2 DEFAULT NULL);
```

Description

Creates an extractor policy. (The first format is for a base policy; the second format is for a policy that is dependent on a base policy.)

Parameters

policy_name
Name of the extractor policy.

extractor
An instance of a subtype of the RDFCTX_EXTRACTOR type that encapsulates the extraction logic for the information extractor.

preferences
Any preferences associated with the policy.

base_policy
Base extractor policy for a dependent policy.

user_models
List of user models for a dependent policy.

user_entailments
List of user entailments for a dependent policy.

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

An extractor policy created using this procedure determines the characteristics of a semantic index that is created using the policy. Each extractor policy refers to an instance of an extractor type, either directly or indirectly. An extractor policy with a direct reference to an extractor type instance can be used to compose other extractor policies that include additional RDF models for ontologies.

An instance of the extractor type assigned to the extractor parameter must be an instance of a direct or indirect subtype of type `mdsys.rdfctx_extractor`.

The RDF models specified in the `user_models` parameter must be accessible to the user that is creating the policy.

The RDF entailments specified in the `user_entailments` parameter must be accessible to the user that is creating the policy. Note that the RDF models underlying the entailments do not get automatically included in the dependent policy. To include one or more of those underlying RDF models, you need to include the models in the `user_models` parameter.

The preferences specified for extractor policy determine the type of repository used for the documents to be indexed and other relevant information. For more information, see Indexing External Documents.

For information about semantic network types and options, see RDF Networks.

Examples

The following example creates an extractor policy using the gatenlp_extractor extractor type, which is included with the Oracle Database support for semantic indexing.

```
begin
  sem_rdfctx.create_policy (policy_name => 'SEM_EXTR',
                            extractor   => mdsys.gatenlp_extractor());
end;
/
```

The following example creates a dependent policy for the previously created extractor policy, and it adds the user-defined RDF model `geo_ontology` to the dependent policy.

```
begin
  sem_rdfctx.create_policy (policy_name => 'SEM_EXTR_PLUS_GEOONT',
                            base_policy => 'SEM_EXTR',
                            user_models => SEM_MODELS ('geo_ontology'));
end;
/
```

17.3 SEM_RDFCTX.DROP_POLICY

Format

```
SEM_RDFCTX.DROP_POLICY(
     policy_name  IN VARCHAR2,
     network_owner IN VARCHAR2 DEFAULT NULL,
     network_name  IN VARCHAR2 DEFAULT NULL);
```

Description

Deletes (drops) an unused extractor policy.

Parameters

policy_name
Name of the extractor policy.

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

An exception is generated if the specified policy being is used for a semantic index for documents or if a dependent extractor policy exists for the specified policy.

For information about semantic network types and options, see RDF Networks.

Examples

The following example drops the `SEM_EXTR_PLUS_GEOONT` extractor policy.

```
begin
  sem_rdfctx.drop_policy (policy_name => 'SSEM_EXTR_PLUS_GEOONT');
end;
/
```

17.4 SEM_RDFCTX.MAINTAIN_TRIPLES

Format

```
SEM_RDFCTX.MAINTAIN_TRIPLES(
     index_name     IN VARCHAR2,
     where_clause   IN VARCHAR2,
     rdfxml_content IN sys.XMLType,
     policy_name    IN VARCHAR2 DEFAULT NULL,
     action         IN VARCHAR2 DEFAULT 'ADD',
     network_owner  IN VARCHAR2 DEFAULT NULL,
     network_name   IN VARCHAR2 DEFAULT NULL);
```

Description

Adds one or more triples to graphs that contain information extracted from specific documents.

Parameters

index_name
Name of the semantic index for documents.

where_clause
A SQL predicate (WHERE clause text without the `WHERE` keyword) on the table in which the documents are stored, to identify the rows for which to maintain the index.

rdfxml_content
Triples, in the form of an RDF/XML document, to be added to the individual graphs corresponding to the documents.

policy_name
Name of the extractor policy. If `policy_name` is null (the default), the triples are added to the information extracted by the default (or the only) extractor policy for the index; if you specify a policy name, the triples are added to the information extracted by that policy.

action
Type of maintenance operation to perform on the triples. The only value currently supported in `ADD` (the default), which adds the triples that are specified in the `rdfxml_content` parameter.

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

The information extracted from the semantically indexed documents may be incomplete and lacking in proper context. This procedure enables a domain expect to add triples to individual graphs pertaining to specific semantically indexed documents, so that all subsequent SEM_CONTAINS queries can consider these triples in their document search criteria.

This procedure accepts the index name and WHERE clause text to identify the specific documents to be annotated with the additional triples. For example, the where_clause might be specified as a simple predicate involving numeric data, such as `'docId IN (1,2,3)'`.

For information about semantic network types and options, see RDF Networks.

Examples

The following example annotates a specific document with the semantic index `ArticleIndex` by adding triples to the corresponding individual graph.

```
begin
  sem_rdfctx.maintain_triples(
     index_name      => 'ArticleIndex',
     where_clause    => 'docid = 15',
     rdfxml_content => sys.xmltype(
      '<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
                xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
                xmlns:pred="http://myorg.com/pred/">
       <rdf:Description rdf:about=" http://newscorp.com/Org/ExampleCorp">
         <pred:hasShortName
               rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
             Example
         </pred:hasShortName>
```

```
     </rdf:Description>
    </rdf:RDF>'));
end;
/
```

17.5 SEM_RDFCTX.SET_DEFAULT_POLICY

Format

```
SEM_RDFCTX.SET_DEFAULT_POLICY(
     index_name    IN VARCHAR2,
     policy_name   IN VARCHAR2,
     network_owner IN VARCHAR2 DEFAULT NULL,
     network_name  IN VARCHAR2 DEFAULT NULL);
```

Description

Sets the default extractor policy for a semantic index that is configured with multiple extractor policies.

Parameters

index_name
Name of the semantic index for documents.

policy_name
Name of the extractor policy to be used as the default extractor policy for the specified semantic index. Must be one of the extractor policies listed in the PARAMETERS clause of the CREATE INDEX statement that created `index_name`.

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

When you create a semantic index for documents, you can specify multiple extractor policies as a space-separated list of names in the PARAMETERS clause of the CREATE INDEX statement. As explained in Semantically Indexing Documents, the first policy from this list is used as the default extractor policy for all SEM_CONTAINS queries that do not identify an extractor policy by name. You can use the SEM_RDFCTX.SET_DEFAULT_POLICY procedure to set a different default policy for the index.

For information about semantic network types and options, see RDF Networks.

Examples

The following example sets `CITY_EXTR` as the default extractor policy for the `ArticleIndex` index.

```
begin
  sem_rdfctx.set_default_policy (index_name => 'ArticleIndex',
                                 policy_name => 'CITY_EXTR');
end;
/
```

17.6 SEM_RDFCTX.SET_EXTRACTOR_PARAM

Format

```
SEM_RDFCTX.SET_EXTRACTOR_PARAM(
     param_key     IN VARCHAR2,
     patam_value   IN VARCHAR2,
     param_desc    IN VARCHAR2,
     network_owner IN VARCHAR2 DEFAULT NULL,
     network_name  IN VARCHAR2 DEFAULT NULL);
```

Description

Configures the Oracle Database semantic indexing support to work with external information extractors, such as Calais and GATE.

Parameters

param_key
Key for the parameter to be set.

param_value
Value for the parameter to be set.

param_desc
Short description for the parameter to be set.

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

To use this procedure, you must be connected as SYSTEM (not SYS … AS SYSDBA) or another non-SYS user with the DBA role.

To work with the Calais extractor type (see Configuring the Calais Extractor type), you must specify values for the following parameters:

- `CALAIS_WS_ENDPOINT`: Web service end point for Calais.
- `CALAIS_KEY`: License key for Calais.
- `CALAIS_WS_SOAPACTION`: SOAP action for the Calais Web service.

To work with the General Architecture for Text Engineering (GATE) extractor type (see Working with General Architecture for Text Engineering (GATE)), you must specify values for the following parameters:

- `GATE_NLP_HOST`: Host for the GATE NLP Listener.
- `GATE_NLP_PORT`: Port for the GATE NLP Listener.

In addition to these parameters, you may need to specify a value for the `HTTP_PROXY` parameter to work with information extractors or index documents that are outside the firewall.

A database instance only has one set of values for these parameters, and they are used for all instances of semantic indexes using the corresponding information extractor. You can use this procedure if you need to change the existing values of any of the parameters.

For information about semantic network types and options, see RDF Networks.

Examples

For examples, see the following sections:

- Configuring the Calais Extractor type
- Working with General Architecture for Text Engineering (GATE)

18
SEM_RDFSA Package Subprograms

The SEM_RDFSA package contains subprograms (functions and procedures) for providing fine-grained access control to RDF data using Oracle Label Security (OLS).

To use the subprograms in this chapter, you should understand the conceptual and usage information in RDF Semantic Graph Overview and Fine-Grained Access Control for RDF Data.

This chapter provides reference information about the subprograms, listed in alphabetical order.

18.1 SEM_RDFSA.APPLY_OLS_POLICY

Format

```
SEM_RDFSA.APPLY_OLS_POLICY(
     policy_name    IN VARCHAR2,
     rdfsa_options  IN NUMBER  DEFAULT SEM_RDFSA.SECURE_SUBJECT,
     table_options  IN VARCHAR2 DEFAULT 'ALL_CONTROL',
     label_function IN VARCHAR2 DEFAULT NULL,
     predicate      IN VARCHAR2 DEFAULT NULL,
     network_owner  IN VARCHAR2 DEFAULT NULL,
     network_name   IN VARCHAR2 DEFAULT NULL);
```

Description

Applies an OLS policy to the semantic data store.

Parameters

policy_name
Name of an existing OLS policy.

rdfsa_options
Options specifying the mode of fine-grained access control to be enabled for RDF data. The default option for securing RDF data involves assigning sensitivity labels for the resources appearing the triples' subject position. You can override the defaults by using the

`rdfsa_options` parameter and specifying one of the constants defined in Table 18-1 in the Usage Notes.

table_options
Policy enforcement options. The default value (`ALL_CONTROL`) is the only supported value for this procedure.

label_function
A string invoking a function to return a label value to use as the default.

predicate
An additional predicate to combine with the label-based predicate.

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

The OLS policy specified with this procedure must be created with CTXT1 as the column name, and it should use default policy options. For information about policy options, see *Oracle Label Security Administrator's Guide*.

This procedure invokes the `sa_policy_admin.apply_table_policy` procedure on multiple tables defined in the semantic network. The parameters `table_options`, `label_function`, and `predicate` for the SEM_RDFSA.APPLY_OLS_POLICY procedure have same semantics as the parameters with same names in the sa_policy_admin.apply_table_policy procedure.

For the `rdfsa_options` parameter, you can specify the package constant for the desired option. Table 18-1 lists these constants and their descriptions.

Table 18-1 SEM_RDFSA Package Constants for rdfsa_options Parameter

Constant	Description
SEM_RDFSA.SECURE_SUBJECT	Assigns sensitivity labels for the resources appearing the triples' subject position.
SEM_RDFSA.SECURE_PREDICATE	Assigns sensitivity labels for the resources appearing the triples' predicate position.
SEM_RDFSA.SECURE_OBJECT	Assigns sensitivity labels for the resources appearing the triples' object position.
SEM_RDFSA.TRIPLE_LEVEL_ONLY	Applies triple-level security. Provides good performance, and eliminates the need to assign labels to individual resources. (Requires that Patch 9819833, available from My Oracle Support, be installed.)
SEM_RDFSA.OPT_DEFINE_BEFORE_USE	Restricts the use of an RDF resource in a triple before the sensitivity label is defined for the resource. If this option is not specified, the user's initial row label is used as the default label for the resource upon first use.
SEM_RDFSA.OPT_RELAX_TRIPLE_LABEL	Relaxes the dominating relationship that exists between the triple label and the labels associated with all its components. With this option, a triple can be defined if the user has READ access to all the triple components and the triple label may not bear any relationship with the component labels. Without this option, the triple label should at least cover the label for all its components.

You can specify a function in the `label_function` parameter to generate custom labels for newly inserted triples. The label function is associated with the RDF_LINK$ table, and the columns in this table may be configured as parameters to the label function as shown in the following example:

```
fgac_admin.new_triple_label(:new.model_id,
                            :new.start_node_id,
                            :new.p_value_id,
                            :new.canon_end_node_id)'
```

Because the OLS policy is applied to more than one table with different structures, the only valid column reference in any predicates assigned to the `predicate` parameter is that of the label column: CTXT1. If OLS is enabled for a semantic data store with existing data, you can specify a predicate of the form `'OR CTXT1 is null'` to be able to continue using this data with no access restrictions.

An OLS-enabled semantic data store uses sensitivity labels for all the RDF triples organized in multiple models. User access to such triples, through model views and SEM_MATCH queries, is restricted by the OLS policy. Additionally, independent of a user owning the semantic model, access to the triple column (of type SDO_RDF_TRIPLE_S) in the RDFT triple view is restricted to users with FULL access privileges with the OLS policy.

The triples are inserted into a specific RDF model using the INSERT privileges on the corresponding RDFT triple view. A sensitivity label for the new triple is generated using the user's session context (initial row label) or the label function. The triple is validated for any RDF policy violations using labels associated with the triple components. Although the triple information may not be accessed through the RDFT triple view, the model view may be queried to access the triples, while enforcing the OLS policy restrictions. If you have the necessary policy privileges (such as writeup, writeacross), you can update the CTXT1 column in the model view to reset the label assigned to the triple. The new label is automatically validated for any RDF policy violations involving the triple components. Update privilege on the CTXT1 column of the model view is granted to the owner of the model, and this user may selectively grant this privilege to other users.

If the RDF models are created in schemas other than the user with FULL access, necessary privileges on the model objects -- specifically, read/write access on the RDFT triple view, read access to the model view, and write access to the CTXT1 column in the model view -- can be granted to such users for maintenance operations. These operations include bulk loading into the model, resetting any sensitivity labels assigned to the triples, and creating entailments using the model.

To disable the OLS policy, use the SEM_RDFSA.DISABLE_OLS_POLICY procedure.

For information about support for OLS, see Fine-Grained Access Control for RDF Data.

For information about semantic network types and options, see RDF Networks.

Examples

The following example enable secure access to RDF data with secure subject and secure predicate options.

```
begin
  sem_rdfsa.apply_ols_policy(
        policy_name   => 'defense',
        rdfsa_options => sem_rdfsa.SECURE_SUBJECT+
                         sem_rdfsa.SECURE_PREDICATE,
        network_owner => 'RDFUSER',
        network_name  => 'NET1');
```

```
end;
/
```

The following example extends the preceding example by specifying a Define Before Use option, which allows a user to define a triple only if the triple components secured (Subject, Predicate or Object) are predefined with an associated sensitivity label. This configuration is effective if the user inserting the triple does not have execute privileges on the SEM_RDFSA package.

```
begin
  sem_rdfsa.apply_ols_policy(
        policy_name   => 'defense',
        rdfsa_options => sem_rdfsa.SECURE_SUBJECT+
                         sem_rdfsa.SECURE_PREDICATE+
                         sem_rdfsa.OPT_DEFINE_BEFORE_USE,
        network_owner => 'RDFUSER',
        network_name  => 'NET1');
end;
/
```

18.2 SEM_RDFSA.DISABLE_OLS_POLICY

Format

```
SEM_RDFSA.DISABLE_OLS_POLICY(
     network_owner IN VARCHAR2 DEFAULT NULL,
     network_name  IN VARCHAR2 DEFAULT NULL);
```

Description

Disables the OLS policy that has been previously applied to or enabled on the semantic data store.

Parameters

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

You can use this procedure to disable temporarily the OLS policy that had been applied to or enabled for the semantic data store. The user disabling the policy should have the necessary privileges to administer OLS policies and should also have access to the OLS policy applied to RDF data.

The sensitivity labels assigned to various RDF resources and triples are preserved and the OLS policy may be re-enabled to enforce them. New resources with specific labels can be added, or labels for existing triples and resources can be updated when the OLS policy is disabled.

To apply an OLS policy, use the SEM_RDFSA.APPLY_OLS_POLICY procedure; to enable an OLS policy that had been disabled, use the SEM_RDFSA.ENABLE_OLS_POLICY procedure.

For information about support for OLS, see Fine-Grained Access Control for RDF Data.

For information about semantic network types and options, see RDF Networks.

Examples

The following example disables the OLS policy for the semantic data store.

```
begin
  sem_rdfsa.disable_ols_policy;
end;
/
```

18.3 SEM_RDFSA.ENABLE_OLS_POLICY

Format

```
SEM_RDFSA.ENABLE_OLS_POLICY(
     network_owner IN VARCHAR2 DEFAULT NULL,
     network_name  IN VARCHAR2 DEFAULT NULL);
```

Description

Enables the OLS policy that has been previously disabled.

Parameters

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

You can use this procedure to enable the OLS policy that had been disabled for the semantic data store. The user enabling the policy should have the necessary privileges to administer OLS policies and should also have access to the OLS policy applied to RDF data.

To disable an OLS policy, use the SEM_RDFSA.DISABLE_OLS_POLICY procedure.

For information about support for OLS, see Fine-Grained Access Control for RDF Data.

For information about support for OLS, see Fine-Grained Access Control for RDF Data.

Examples

The following example enables the OLS policy for the semantic data store.

```
begin
  sem_rdfsa.enable_ols_policy;
end;
/
```

18.4 SEM_RDFSA.REMOVE_OLS_POLICY

Format

```
SEM_RDFSA.REMOVE_OLS_POLICY(
     network_owner IN VARCHAR2 DEFAULT NULL,
     network_name  IN VARCHAR2 DEFAULT NULL);
```

Description

Permanently removes or detaches the OLS policy from the semantic data store.

Parameters

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

You should have the necessary privileges to administer OLS policies, and you should also have access to the OLS policy applied to RDF data. Once the OLS policy is detached from the semantic data store, all the sensitivity labels previously assigned to the triples and resources are lost.

This operation drops objects that are specifically created to maintain the RDF security policies.

To apply an OLS policy, use the SEM_RDFSA.APPLY_OLS_POLICY procedure.

For information about support for OLS, see Fine-Grained Access Control for RDF Data.

For information about semantic network types and options, see RDF Networks.

Examples

The following example removes the OLS policy that had been previously applied to the semantic data store.

```
begin
  sem_rdfsa.remove_ols_policy;
end;
/
```

18.5 SEM_RDFSA.RESET_MODEL_LABELS

Format

```
SEM_RDFSA.RESET_MODEL_LABELS(
     model_name    IN VARCHAR2,
     network_owner IN VARCHAR2 DEFAULT NULL,
     network_name  IN VARCHAR2 DEFAULT NULL);
```

Description

Resets the labels associated with a model or with global resources; requires that the associated model or models be empty.

Parameters

model_name
Name of the model for which the labels should be reset, or the string `RDF$GLOBAL` to reset the labels associated with all global resources.

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

If you specify a model name, the model must be empty. If you specify `RDF$GLOBAL`, all the models must be empty (that is, no triples in the RDF repository).

You must have FULL access privilege with the OLS policy applied to the semantic data store.

For information about support for OLS, see Fine-Grained Access Control for RDF Data.

For information about semantic network types and options, see RDF Networks.

Examples

The following example removes all resources and their labels associated with the `Contracts` model.

```
begin
   sem_rdfsa.reset_model_labels(model_name => 'Contracts');
end;
/
```

18.6 SEM_RDFSA.SET_PREDICATE_LABEL

Format

```
SEM_RDFSA.SET_PREDICATE_LABEL(
     model_name    IN VARCHAR2,
     predicate     IN VARCHAR2,
     label_string  IN VARCHAR2,
     network_owner IN VARCHAR2 DEFAULT NULL,
     network_name  IN VARCHAR2 DEFAULT NULL);
```

Description

Sets a sensitivity label for a predicate at the model level or for the whole repository.

Parameters

model_name
Name of the model to which the predicate belongs, or the string `RDF$GLOBAL` if the same label should applied for the use of the predicate in all models.

predicate
Predicate for which the label should be assigned.

label_string
OLS row label in string representation.

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

If you specify a model name, you must have read access to the model and execute privileges on the SEM_RDFSA package to perform this operation. If you specify `RDF$GLOBAL`, you must have FULL access privilege with the OLS policy applied to RDF data.

You must have access to the specified label and OLS policy privilege to overwrite an existing label if a label already exists for the predicate. The SECURE_PREDICATE option must be enabled for RDF data.

If an existing predicate label is updated with this operation, the labels for the triples using this predicate must all dominate the new predicate label. The only exception is when the OPT_RELAX_TRIPLE_LABEL option is chosen for the OLS-enabled RDF data.

If you specify `RDF$GLOBAL`, a global predicate with a unique sensitivity label across models is created. If the same predicate is previously defined in one or more models, the global label dominates all such labels and the model-specific labels are replaced for the given predicate.

After a label for a predicate is set, new triples with the predicate can be added only if the triple label (which may be initialized from user's initial row label or using a label function) dominates the predicate's sensitivity label. This dominance relationship can be relaxed with the OPT_RELAX_TRIPLE_LABEL option, in which case the user should at least have read access to the predicate to be able to define a new triple using the predicate.

For information about support for OLS, see Fine-Grained Access Control for RDF Data.

For information about semantic network types and options, see RDF Networks.

Examples

The following example sets a predicate label for `Contracts` model and another predicate label for all models in the database instance.

```
begin
  sem_rdfsa.set_predicate_label(
         model_name   => 'contracts',
         predicate    => '<http://www.myorg.com/pred/hasContractValue>',
         label_string => 'TS:US_SPCL');
end;
/

begin
  sem_rdfsa.set_predicate_label(
         model_name   => 'rdf$global',
         predicate    => '<http://www.myorg.com/pred/hasStatus>',
         label_string => 'SE:US_SPCL:US');
end;
/
```

18.7 SEM_RDFSA.SET_RDFS_LABEL

Format

```
SEM_RDFSA.SET_RDFS_LABEL(
     label_string  IN VARCHAR2,
     inf_override  IN VARCHAR2,
```

```
     network_owner IN VARCHAR2 DEFAULT NULL,
     network_name  IN VARCHAR2 DEFAULT NULL);
```

Description

Sets a sensitivity label for RDFS schema elements.

Parameters

label_string
OLS row label in string representation, to be used as the sensitivity label for all RDF schema constructs.

inf_override
OLS row label to be used as the override for generating labels for inferred triples.

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

This procedure sets or resets the sensitivity label associated with the RDF schema resources, often recognized by `http://www.w3.org/1999/02/22-rdf-syntax-ns#` and `http://www.w3.org/2000/01/rdf-schema#` prefixes for their URIs. You can assign a sensitivity label with restricted access to these resources, so that operations such as creating new RDF classes and adding new properties can be restricted to users with higher privileges.

You must have FULL access privilege with policy applied to RDF data.

RDF schema elements implicitly use the relaxed triple label option, so that the triples using RDFS and OWL constructs for subject, predicate, or object are not forced to have a sensitivity label that dominates the labels associated with the schema constructs. Therefore, a user capable of defining new RDF classes and properties must least have read access to the schema elements.

When RDF schema elements are referred to in the inferred triples, the system-defined and custom label generators consider the inference override label in determining the appropriate label for the inferred triples. If a custom label generator is used, this override label is passed instead of the actual label when an RDF schema element is involved.

For information about support for OLS, see Fine-Grained Access Control for RDF Data.

For information about semantic network types and options, see RDF Networks.

Examples

The following example sets a label with a unique compartment for all RDF schema elements. A user capable of defining new RDF classes and properties is expected to have an exclusive membership to the compartment.

```
begin
  sem_rdfsa.set_rdfs_label(
         label_string  => 'SE:RDFS:',
         inf_override  => 'SE:US_SPCL:US');
end;
/
```

18.8 SEM_RDFSA.SET_RESOURCE_LABEL

Format

```
SEM_RDFSA.SET_RESOURCE_LABEL(
     model_name   IN VARCHAR2,
     resource_uri IN VARCHAR2,
     label_string IN VARCHAR2,
     resource_pos IN VARCHAR2 DEFAULT 'S',
     network_owner IN VARCHAR2 DEFAULT NULL,
     network_name  IN VARCHAR2 DEFAULT NULL);
```

Description

Sets a sensitivity label for a resource that may be used in the subject and/or object position of a triple.

Parameters

model_name
Name of the model to which the resource belongs, or the string `RDF$GLOBAL` if the same label should applied for using the resource in all models.

resource_uri
URI for the resource that may be used as subject or object in one or more triples.

label_string
OLS row label in string representation.

resource_pos
Position of the resource within a triple: `S`, `O`, or `S,O`. You can specify up to two separate labels for the same resource, one to be considered when the resource is used in the subject position of a triple and the other to be considered when it appears in the object position. The values '`S`', '`O`' or '`S,O`' set a label for the resource in subject, object or both subject and object positions, respectively.

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

If you specify a model name, you must have read access to the model and execute privileges on the SEM_RDFSA package to perform this operation. If you specify `RDF$GLOBAL`, you must have FULL access privilege with the OLS policy applied to RDF data.

You must have access to the specified label and OLS policy privilege to overwrite an existing label if a label already exists for the predicate. The SECURE_PREDICATE option must be enabled for RDF data.

If an existing resource label is updated with this operation, the labels for the triples using this resource in the specified position must all dominate the new resource label. The only exception is when the OPT_RELAX_TRIPLE_LABEL option is chosen for the OLS-enabled RDF data.

If you specify `RDF$GLOBAL`, a global resource with a unique sensitivity label across models is created. If the same resource is previously defined in one or more models with the same triple position, the global label dominates all such labels and the model-specific labels are replaced for the given resource in that position.

After a label for a predicate is set, new triples using the resource in the specified position can be added only if the triple label dominates the resource's sensitivity label. This dominance relationship can be relaxed with OPT_RELAX_TRIPLE_LABEL option, in which case, the user should at least have read access to the resource.

For information about support for OLS, see Fine-Grained Access Control for RDF Data.

For information about semantic network types and options, see RDF Networks.

Examples

The following example sets sensitivity labels for multiple resources based on their position.

```
begin
  sem_rdfsa.set_resource_label(
         model_name   => 'contracts',
         resource_uri => '<http://www.myorg.com/contract/projectHLS>',
         label_string => 'SE:US_SPCL:US',
         resource_pos => 'S,O');
end;
/

begin
  sem_rdfsa.set_resource_label(
       model_name   => 'rdf$global',
       resource_uri => '<http://www.myorg.com/contract/status/Complete>',
       label_string => 'SE:US_SPCL:US',
       resource_pos => 'O');
end;
/
```

18.9 SEM_RDFSA.SET_RULE_LABEL

Format

```
SEM_RDFSA.SET_RULE_LABEL(
     rule_base     IN VARCHAR2,
     rule_name     IN VARCHAR2,
     label_string  IN VARCHAR2,
     network_owner IN VARCHAR2 DEFAULT NULL,
     network_name  IN VARCHAR2 DEFAULT NULL);
```

Description

Sets sensitivity label for a rule belonging to a rulebase.

Parameters

rule_base
Name of an existing RDF rulebase.

rule_name
Name of the rule belonging to the rulebase.

label_string
OLS row label in string representation.

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

The sensitivity label assigned to the rule is used to generate the label for the inferred triples when an appropriate label generator option is chosen.

You must have access have access to the rulebase, and you must have FULL access privilege with the OLS policy can assign labels for system-defined rules in the RDFS rulebase.

There is no support for labels assigned to user-defined rules.

For information about support for OLS, see Fine-Grained Access Control for RDF Data.

For information about semantic network types and options, see RDF Networks.

Examples

The following example assigns a sensitivity label for an RDFS rule.

```
begin
sem_rdfsa.set_rule_label (rule_base    => 'RDFS',
                          rule_name    => 'RDF-AXIOMS',
                          label_string => 'SE:US_SPCL:');
end;
/
```

Part IV

Appendixes

The following appendixes are included.

A

Enabling, Downgrading, or Removing RDF Graph Support

You must perform certain steps before you can use any types, synonyms, or PL/SQL packages related to RDF Graph support in the current Oracle Database release.

You must run one or more scripts, and you must ensure that Oracle Spatial is installed and the Partitioning option is enabled. These requirements are explained in Enabling RDF Graph Support and its related subtopics.

This appendix also describes the steps if, after enabling RDF Graph support, you need to do any of the following:

- Downgrade the RDF Graph support to that provided with a previous Oracle Database release, as explained in Downgrading RDF Graph Support to a Previous Release.
- Remove all support for RDF Graph from the database, as explained in Removing RDF Graph Support.

A.3 Removing RDF Graph Support

You can remove the RDF Graph support from the database.

However, removing this support is ***strongly discouraged***, unless you have a solid reason for doing it. After you remove this support, no applications or database users will be able to use any types, synonyms, or PL/SQL packages related to RDF Graph support.

To remove the RDF Graph support from the database, perform the following steps:

1. Connect to the database as the SYS user with SYSDBA privileges (SYS AS SYSDBA, and enter the SYS account password when prompted).
2. Start SQL*Plus, and enter the following statement:
   - Linux: `@$ORACLE_HOME/md/admin/semremov.sql`
   - Windows: `@%ORACLE_HOME%\md\admin\semremov.sql`

> **Note:**
>
> If you are in a multitenant environment, run the script with `catcon.pl`. See "Running Oracle-Supplied SQL Scripts in a CDB" in Oracle Database Administrator's Guide.

The `semremov.sql` script drops the semantic network and removes any RDF Graph types, tables, and PL/SQL packages.

A.1 Enabling RDF Graph Support

Before you can use any types, synonyms, or PL/SQL packages related to RDF Graph support in the current Oracle Database release, you must either install the capabilities in a new Oracle Database installation or upgrade the capabilities from a previous release.

Install of RDF Graph support is included in install of Oracle Spatial. So you must ensure that Oracle Spatial is installed. In addition, Partitioning must be enabled. Restricted use of Partitioning is allowed free of charge for supporting Graph feature of Oracle Database. See Restricted Use Licenses for more information.

- Enabling RDF Semantic Graph Support in a New Database Installation
- Upgrading RDF Semantic Graph Support from Release 11.1, 11.2, or 12.1
- Workspace Manager and Virtual Private Database Desupport

A.1.1 Enabling RDF Semantic Graph Support in a New Database Installation

RDF Semantic Graph is automatically enabled when Oracle Spatial Release 12.2 or later is installed. See Manually Installing Spatial if you do not have Oracle Spatial installed by default at the time of Oracle Database installation.

If RDF Semantic Graph was enabled successfully, a row with the following column values will exist in the MDSYS.RDF_PARAMETER table:

- NAMESPACE: `MDSYS`
- ATTRIBUTE: `SEM_VERSION`
- VALUE: (string starting with `12.2`)
- DESCRIPTION: `VALID`

A.1.2 Upgrading RDF Semantic Graph Support from Release 11.1, 11.2, or 12.1

If you are upgrading from Oracle Database Release 11.1 or 11.2 that includes the semantic technologies support, the semantic technologies support is automatically upgraded to Release 12.1 or later when the database is upgraded.

However, you may also need to migrate RDF data if you have an existing Release 11.1 or 11.2 RDF network containing triples that include typed literal values of type xsd:float, xsd:double, xsd:boolean, or xsd:time.

To check if you need to migrate RDF data, connect to the database as a user with DBA privileges and query the MDSYS.RDF_PARAMETER table, as follows:

```
SELECT namespace, attribute, value FROM mdsys.rdf_parameter
  WHERE namespace='MDSYS'
  AND attribute IN ('FLOAT_DOUBLE_DECIMAL',
                    'XSD_TIME', 'XSD_BOOLEAN',
                    'DATA_CONVERSION_CHECK');
```

If the FLOAT_DOUBLE_DECIMAL, XSD_TIME, or XSD_BOOLEAN attributes have the string value `INVALID` or if the DATA_CONVERSION_CHECK attribute has the string value `FAILED_UNABLE_TO_LOCK_APPLICATION_TABLES`, `FAILED_INSUFFICIENT_WORKSPACE_PRIVILEGES`, or `FAILED_OLS_POLICIES_ARE_ENABLED`, you need to migrate RDF data.

However, if the FLOAT_DOUBLE_DECIMAL, XSD_TIME, and XSD_BOOLEAN attributes do not exist or have the string value `VALID` and if the DATA_CONVERSION_CHECK attribute does not exist, you do *not* need to migrate RDF data. However, if your semantic network may have any empty RDF literals, see Handling of Empty RDF Literals; and if you choose to migrate existing empty literals to the new format, follow the steps in this section.

To migrate RDF data, follow these steps:

1. Connect to the database as the SYSTEM (not SYS .. AS SYSDBA) user or another non-SYS user with the DBA role, and enter: `SET CURRENT_SCHEMA=MDSYS`
2. Ensure that the user MDSYS has the following privileges:
   - INSERT privilege on all application tables in the semantic network
   - ALTER ANY INDEX privilege (optional: only necessary if Semantic Indexing for Documents is being used)
   - ACCESS privilege for any workspace in which version-enabled application tables have been modified (optional: only necessary if Workspace Manager is being used for RDF data)
3. Ensure that any OLS policies for RDF data are temporarily disabled (optional: only necessary if OLS for RDF Data is being used). OLS policies can be re-enabled after running convert_old_rdf_data.
4. Start SQL*Plus. If you want to migrate the RDF data without converting existing empty literals to the new format (see Handling of Empty RDF Literals), enter the following statement:

   ```
   EXECUTE sdo_rdf_internal.convert_old_rdf_data;
   ```

   If you want to migrate the RDF data and also convert existing empty literals to the new format, call `convert_old_rdf_data` with the `flags` parameter set to `'CONVERT_ORARDF_NULL'`. In addition, you can use an optional `tablespace_name` parameter to specify the tablespace to use when creating intermediate tables during data migration. For example, the following statement migrates old semantic data, converts existing `"orardf:null "` values to "", and uses the MY_TBS tablespace for any intermediate tables:

   ```
   EXECUTE sdo_rdf_internal.convert_old_rdf_data(
     flags=>'CONVERT_ORARDF_NULL',
     tablespace_name=>'MY_TBS');
   ```

   The `sdo_rdf_internal.convert_old_rdf_data` procedure may take a significant amount of time to run if the semantic network contains many triples that are using (or affected by use of) xsd:float, xsd:double, xsd:time, or xsd:boolean typed literals.

5. Connect to the database as the SYS user with SYSDBA privileges (SYS AS SYSDBA, and enter the SYS account password when prompted). Then enter the following statement:
   - Linux: `@$ORACLE_HOME/md/admin/semrelod.sql`
   - Windows: `@%ORACLE_HOME%\md\admin\semrelod.sql`

> **Note:**
>
> You may encounter the ORA-00904 (invalid identifier) error when executing a SEM_MATCH query if the `sdo_rdf_internal.convert_old_rdf_data` procedure and the `semrelod.sql` script were not run after the upgrade to Release 12.1 or later.

- Required Data Migration After Upgrade
- Handling of Empty RDF Literals

A.1.2.1 Required Data Migration After Upgrade

After the database upgrade completes, if you have existing RDF data from a previous release, you must migrate the RDF data. If you do not perform the data migration, you will encounter the following error when running SEM_MATCH queries:

```
ORA-20000:  RDF_VALUE$ Table needs data migration with SEM_APIS.MIGRATE_DATA_TO_CURRENT
```

Columns were added to the MDSYS.RDF_VALUE$ table in Release 12.2 (see Enhanced RDF ORDER BY Query Processing). These columns must be populated after upgrading an existing RDF network. The need for migration will be noted with the following row in the MDSYS.RDF_PARAMETER table:

- NAMESPACE: `MDSYS`
- ATTRIBUTE: `RDF_VALUE$`
- VALUE: `INVALID_ORDER_COLUMNS`
- DESCRIPTION: `RDF_VALUE$ Table needs data migration with SEM_APIS.MIGRATE_DATA_TO_CURRENT`

If migration is needed, the RDF Semantic Graph installation will initially be marked as `INVALD`, which is signified with the following row in MDSYS.RDF_PARAMETER:

- NAMESPACE: `MDSYS`
- ATTRIBUTE: `SEM_VERSION`
- VALUE: (string starting with `12.2`)
- DESCRIPTION: `INVALID`

To perform data migration by populating new MDSYS.RDF_VALUE$ columns, follow these steps:

1. 1. Connect to the database as the SYSTEM (not SYS .. AS SYSDBA) user or as another non-SYS user with the DBA role.
2. Run the following statement:

```
EXECUTE sem_apis.migrate_data_to_current;
```

If data migration was successful, the INVALID_ORDER_COLUMNS row will be removed from MDSYS.RDF_PARAMETER and the SEM_VERSION row will have a DESCRIPTION value of `VALID`.

Moreover, additional data migration may be required if you are upgrading an existing Release 11.1 or 11.2 RDF network containing triples that include typed literal values of type xsd:float, xsd:double, xsd:boolean, or xsd:time.

To check if you need to perform this additional RDF data migration, connect to the database as a user with DBA privileges and query the MDSYS.RDF_PARAMETER table, as follows:

```
SELECT namespace, attribute, value FROM mdsys.rdf_parameter
  WHERE namespace='MDSYS'
  AND attribute IN ('FLOAT_DOUBLE_DECIMAL',
                    'XSD_TIME', 'XSD_BOOLEAN',
                    'DATA_CONVERSION_CHECK');
```

If the FLOAT_DOUBLE_DECIMAL, XSD_TIME, or XSD_BOOLEAN attributes have the string value `INVALID` or if the DATA_CONVERSION_CHECK attribute has the string value `FAILED_UNABLE_TO_LOCK_APPLICATION_TABLES`, `FAILED_INSUFFICIENT_WORKSPACE_PRIVILEGES`, or `FAILED_OLS_POLICIES_ARE_ENABLED`, you need to migrate RDF data.

However, if the FLOAT_DOUBLE_DECIMAL, XSD_TIME, and XSD_BOOLEAN attributes do not exist or have the string value `VALID` and if the DATA_CONVERSION_CHECK attribute does not exist, you do *not* need to migrate RDF data. However, if your semantic network may have any empty RDF literals, see Handling of Empty RDF Literals; and if you choose to migrate existing empty literals to the new format, follow the steps in this section.

To migrate the RDF data, follow these steps:

1. Connect to the database as the SYSTEM (not SYS .. AS SYSDBA) user or as another non-SYS user with the DBA role , and enter: `SET CURRENT_SCHEMA=MDSYS`
2. Ensure that the user MDSYS has the following privileges:
   - INSERT privilege on all application tables in the semantic network
   - ALTER ANY INDEX privilege (optional: only necessary if Semantic Indexing for Documents is being used)
   - ACCESS privilege for any workspace in which version-enabled application tables have been modified (optional: only necessary if Workspace Manager is being used for RDF data)
3. Ensure that any OLS policies for RDF data are temporarily disabled (optional: only necessary if OLS for RDF Data is being used). OLS policies can be re-enabled after running convert_old_rdf_data.
4. Start SQL*Plus. If you want to migrate the RDF data without converting existing empty literals to the new format (see Handling of Empty RDF Literals), enter the following statement:

   ```
   EXECUTE sdo_rdf_internal.convert_old_rdf_data;
   ```

   If you want to migrate the RDF data and also convert existing empty literals to the new format, call `convert_old_rdf_data` with the `flags` parameter set to `'CONVERT_ORARDF_NULL'`. In addition, you can use an optional `tablespace_name` parameter to specify the tablespace to use when creating intermediate tables during data migration. For example, the following statement migrates old semantic data, converts existing `"orardf:null "` values to "", and uses the MY_TBS tablespace for any intermediate tables:

```
EXECUTE sdo_rdf_internal.convert_old_rdf_data(
  flags=>'CONVERT_ORARDF_NULL',
  tablespace_name=>'MY_TBS');
```

The `sdo_rdf_internal.convert_old_rdf_data` procedure may take a significant amount of time to run if the semantic network contains many triples that are using (or affected by use of) xsd:float, xsd:double, xsd:time, or xsd:boolean typed literals.

5. Connect to the database as the SYS user with SYSDBA privileges (SYS AS SYSDBA), and enter the SYS account password when prompted). Then enter the following statement:
   - Linux: `@$ORACLE_HOME/md/admin/semrelod.sql`
   - Windows: `@%ORACLE_HOME%\md\admin\semrelod.sql`

> **Note:**
>
> You may encounter the ORA-00904 (invalid identifier) error when executing a SEM_MATCH query if the `sdo_rdf_internal.convert_old_rdf_data` procedure and the `semrelod.sql` script were not run after the upgrade to Release 12.1 or later.

A.1.2.2 Handling of Empty RDF Literals

The way empty-valued RDF literals are handled was changed in Release 11.2. Before this release, the values of empty-valued literals were converted to "`orardf:null`". In Release 11.2 and later, such values are stored without modification (that is, as ""). However, whether you migrate existing "`orardf:null`" values to "" is optional.

To check if "`orardf:null`" values exist in your semantic network, connect to the database as a user with DBA privileges and query the MDSYS.RDF_PARAMETER table, as follows:

```
SELECT namespace, attribute, value FROM mdsys.rdf_parameter
  WHERE namespace='MDSYS'
  AND attribute = 'NULL_LITERAL';
```

If the NULL_LITERAL attribute has the value `EXISTS`, then "`orardf:null`" values are present in your semantic network.

A.1.3 Workspace Manager and Virtual Private Database Desupport

Effective with Oracle Database Release 12.2, the following are no longer supported:

- Workspace Manager support for RDF data
- Virtual Private Database (VPD) support for RDF data

If an existing semantic network that contains Workspace Manager (WM) or Virtual Private Database (VPD) data is upgraded, the RDF Semantic Graph installation will be marked as `INVALID`. In addition, the MDSYS.RDF_PARAMETER table will contain a row with description `Feature not supported in current version' for the unsupported component`. To correct this situation, all existing WM and VPD data should be dropped, and the WM and VPD components should be uninstalled.

To uninstall Workspace Manager support for RDF data:

1. Connect to the database as the SYS user with SYSDBA privileges (SYS AS SYSDBA, and enter the SYS account password when prompted).

2. Start SQL*Plus, and enter the following statement:
   - Linux: `@$ORACLE_HOME/md/admin/sdordfwm_rm.sql`
   - Windows: `@%ORACLE_HOME%\md\admin\sdordfwm_rm.sql`

> **Note:**
>
> If you are in a multitenant environment, run the script with `catcon.pl`. See “Running Oracle-Supplied SQL Scripts in a CDB” in Oracle Database Administrator’s Guide.

To uninstall Virtual Private Database support for RDF data:

1. Connect to the database as the SYSTEM user (not SYS … AS SYSDBA) or as another non-SYS user with the DBA role.
2. Start SQL*Plus, and enter the following statement:

```
EXECUTE mdsys.sem_rdfsa_dr.uninstall_vpd;
```

After performing the necessary uninstall operations, reset the network validity as follows:

1. Connect to the database as the SYS user with SYSDBA privileges (SYS AS SYSDBA, and enter the SYS account password when prompted).
2. Start SQL*Plus, and enter the following statement:
   - Linux: `@$ORACLE_HOME/md/admin/semvalidate.sql`
   - Windows: `@%ORACLE_HOME%\md\admin\semvalidate.sql`

> **Note:**
>
> If you are in a multitenant environment, run the script with `catcon.pl`. See “Running Oracle-Supplied SQL Scripts in a CDB” in Oracle Database Administrator’s Guide.

A.2 Downgrading RDF Graph Support to a Previous Release

You can downgrade the RDF Graph support, in conjunction with an Oracle Database downgrade to Release 12.1.

However, downgrading is ***strongly discouraged***, except for rare cases where it is necessary. If you downgrade to a previous release, you will not benefit from bug fixes and enhancements that have been made in intervening releases.

- Downgrading to Release 12.1 Semantic Graph Support

A.2.1 Downgrading to Release 12.1 Semantic Graph Support

If you need to downgrade to Oracle Database Release 12.1, the RDF semantic graph support component will be downgraded automatically when you downgrade the database. However, any RDF or OWL data that is specific to Release 12.2 (that is, Release 12.2 or later RDF/OWL persistent structures that are not supported in previous versions) must be dropped *before* you perform the downgrade, so that the database is compatible with Release 12.1.

To check if any Release 12.2 or later RDF data is incompatible with Release 12.1, perform the following steps:

1. Connect to the database (Release 12.2 or later) as the SYS user with SYSDBA privileges (SYS AS SYSDBA, and enter the SYS account password when prompted).
2. Start SQL*Plus, and enter the following statements:

```
SET SERVEROUT ON
EXECUTE SDO_SEM_DOWNGRADE.CHECK_121_COMPATIBLE;
```

If any RDF data is incompatible with Release 12.1, the procedure generates an error and displays a list of the incompatible data. In this case, you must perform the following steps:

1. Remove any Release 12.2 or later release-specific RDF or OWL data if you have not already done so, as explained earlier in this section.
2. Perform the database downgrade.
3. Connect to the Release 12.1 database as the SYS user with SYSDBA privileges (SYS AS SYSDBA, and enter the SYS account password when prompted).
4. Start SQL*Plus, and enter the following statement:
   - Linux: `@$ORACLE_HOME/md/admin/catsem.sql`
   - Windows: `@%ORACLE_HOME%\md\admin\catsem.sql`

   > **Note:**
   >
   > If you are in a multitenant environment, run the script with `catcon.pl`. See “Running Oracle-Supplied SQL Scripts in a CDB” in Oracle Database Administrator’s Guide.

   If the script completes successfully, a row with the following column values is inserted into the MDSYS.RDF_PARAMETER table:
   - NAMESPACE: `MDSYS`
   - ATTRIBUTE: `SEM_VERSION`
   - VALUE: (string starting with `12.1`)
   - DESCRIPTION: `VALID`

   After the `catsem.sql` script completes successfully, Oracle semantic technologies support for Release 11.2 is enabled and ready to use, and all Release 12.1-compatible data is preserved.

B

SEM_MATCH Support for Spatial Queries

This appendix provides reference information for SPARQL extension functions for performing spatial queries in SEM_MATCH.

> **Note:**
>
> The SEM_MATCH table function is supported only if Oracle JVM is enabled on your Oracle Autonomous Database Serverless deployments. To enable Oracle JVM, see Use Oracle Java in *Using Oracle Autonomous Database Serverless* for more information.

To use these functions, you must understand the concepts explained in Spatial Support.

> **Note:**
>
> Throughout this appendix `geomLiteral` is used as a placeholder for `orageo:WKTLiteral`, `ogc:wktLiteral`, and `ogc:gmlLiteral`, which can be used interchangeably, in format representations and parameter descriptions. (However, `orageo:WKTLiteral` or `ogc:wktLiteral` is used in actual examples.)

This appendix includes the GeoSPARQL and Oracle-specific functions which are explained in the following sections:

- GeoSPARQL Functions for Spatial Support
- Oracle-Specific Functions for Spatial Support

B.1 GeoSPARQL Functions for Spatial Support

This section provides reference information about the GeoSPARQL functions:

- ogcf:aggBoundingBox
- ogcf:aggBoundingCircle
- ogcf:aggCentroid
- ogcf:aggConcaveHull
- ogcf:aggConvexHull
- ogcf:aggUnion
- ogcf:Area
- ogcf:asGeoJSON
- ogcf:asGML
- ogcf:asKML

- ogcf:asWKT
- ogcf:boundary
- ogcf:boundingCircle
- ogcf:buffer
- ogcf:concaveHull
- ogcf:convexHull
- ogcf:coordinateDimension
- ogcf:difference
- ogcf:dimension
- ogcf:distance
- ogcf:envelope
- ogcf:geometryN
- ogcf:geometryType
- ogcf:getSRID
- ogcf:intersection
- ogcf:is3D
- ogcf:isEmpty
- ogcf:isMeasured
- ogcf:isSimple
- ogcf:length
- ogcf:maxX
- ogcf:maxY
- ogcf:maxZ
- ogcf:metricArea
- ogcf:metricBuffer
- ogcf:metricLength
- ogcf:metricPerimeter
- ogcf:minX
- ogcf:minY
- ogcf:minZ
- ogcf:numGeometries
- ogcf:perimeter
- ogcf:relate
- ogcf:sfContains
- ogcf:sfCrosses
- ogcf:sfDisjoint
- ogcf:sfEquals
- ogcf:sfIntersects

- ogcf:sfOverlaps
- ogcf:sfTouches
- ogcf:sfWithin
- ogcf:spatialDimension
- ogcf:symDifference
- ogcf:transform
- ogcf:union

B.1.1 ogcf:aggBoundingBox

Format

ogcf:aggBoundingBox(geom : geomLiteral) : ogc:wktLiteral

Description

Aggregate that returns a single geometry object that is the minimum bounding box (rectangle) of the input set of geometries.

Parameters

geom
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example finds the aggregate bounding box of U.S. Congressional district polygons.

```
SELECT bb
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT (ogcf:aggBoundingBox(?cgeom) AS ?bb)
 WHERE
 { ?cdist orageo:hasExactGeometry ?cgeom }'
,sem_models('gov_all_vm'), null
, null ,null, null, ' ALLOW_DUP=T '));
```

B.1.2 ogcf:aggBoundingCircle

Format

ogcf:aggBoundingCircle(geom : geomLiteral) : ogc:wktLiteral

Description

Aggregate that returns a single geometry object that is the minimum bounding circle of the input set of geometries.

Parameters

geom
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example finds the aggregate bounding circle of U.S. Congressional district polygons.

```
SELECT bc
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT (ogcf:aggBoundingCircle(?cgeom) AS ?bc)
 WHERE
 { ?cdist orageo:hasExactGeometry ?cgeom }'
,sem_models('gov_all_vm'), null
, null ,null, null, ' ALLOW_DUP=T '));
```

B.1.3 ogcf:aggCentroid

Format

ogcf:aggCentroid(geom : geomLiteral) : ogc:wktLiteral

Description

Aggregate that returns a single geometry object that is the centroid (center point) of the input set of geometries.

Parameters

geom
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example finds the aggregate centroid of U.S. Congressional district polygons.

```
SELECT c
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT (ogcf:aggCentroid(?cgeom) AS ?c)
 WHERE
 { ?cdist orageo:hasExactGeometry ?cgeom }'
,sem_models('gov_all_vm'), null
, null ,null, null, ' ALLOW_DUP=T '));
```

B.1.4 ogcf:aggConcaveHull

Format

ogcf:aggConcaveHull(geom : geomLiteral) : ogc:wktLiteral

Description

Aggregate that returns a single geometry object that is the concave hull of the input set of geometries. The concave hull is a polygon that represents the area of the input geometry, such as a collection of points. With complex input geometries, the concave hull is typically significantly smaller in area than the convex hull.

Parameters

geom
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example finds the aggregate concave hull of U.S. Congressional district polygons.

```
SELECT ch
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT (ogcf:aggConcaveHull(?cgeom) AS ?ch)
 WHERE
 { ?cdist orageo:hasExactGeometry ?cgeom }'
,sem_models('gov_all_vm'), null
, null ,null, null, ' ALLOW_DUP=T '));
```

B.1.5 ogcf:aggConvexHull

Format

ogcf:aggConvexHull(geom : geomLiteral) : ogc:wktLiteral

Description

Aggregate that returns a single geometry object that is the convex hull of the input set of geometries. The convex hull is a simple convex polygon that completely encloses the geometry object, using as few straight-line sides as possible to create the smallest polygon that completely encloses the geometry object.

Parameters

geom
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example finds the aggregate convex hull of U.S. Congressional district polygons.

```
SELECT ch
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT (ogcf:aggConvexHull(?cgeom) AS ?ch)
 WHERE
```

```
 { ?cdist orageo:hasExactGeometry ?cgeom }'
,sem_models('gov_all_vm'), null
, null ,null, null, ' ALLOW_DUP=T '));
```

B.1.6 ogcf:aggUnion

Format

ogcf:aggUnion(geom : geomLiteral) : ogc:wktLiteral

Description

Aggregate that returns a single geometry object that is the topological union of the input set of geometries.

Parameters

geom
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example finds the aggregate union of U.S. Congressional district polygons.

```
SELECT u
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT (ogcf:aggUnion(?cgeom) AS ?u)
 WHERE
 { ?cdist orageo:hasExactGeometry ?cgeom }'
,sem_models('gov_all_vm'), null
, null ,null, null, ' ALLOW_DUP=T '));
```

B.1.7 ogcf:Area

Format

ogcf:area(geom : geomLiteral, units : xsd:anyURI) : ogc:wktLiteral

Description

Returns the area of a two-dimensional polygon.

Parameters

geom
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

units
Unit of measurement:

- A URI of the form `<http://xmlns.oracle.com/rdf/geo/uom/{SDO_UNIT}>` (for example, `<http://xmlns.oracle.com/rdf/geo/uom/M>`). Any SDO_UNIT value from the MDSYS.SDO_AREA_UNITS table will be recognized. See the section about Unit Of Measurement Support in *Oracle Spatial Developer's Guide* for more information about unit of measurement specification.
- A URI from the QUDT vocabulary of units that has an equivalent unit in MDSYS.SDO_AREA_UNITS table. For example, `<http://qudt.org/vocab/unit/M2>` for square meter.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example finds the area in square meters of each U.S. Congressional district polygon.

```
SELECT name, ca
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT ?name (ogcf:area(?cgeom, <http://qudt.org/vocab/unit/M2>) AS ?ca)
 WHERE
 { ?person usgovt:name ?name .
   ?person pol:hasRole ?role .
   ?role pol:forOffice ?office .
   ?office pol:represents ?cdist .
   ?cdist orageo:hasExactGeometry ?cgeom }'
,sem_models('gov_all_vm'), null
, null ,null, null, ' ALLOW_DUP=T '));
```

B.1.8 ogcf:asGeoJSON

Format

ogcf:asGeoJSON(geom : geomLiteral) : ogc:geoJSONLiteral

Description

Converts geom to an equivalent GeoJSON representation.

Parameters

geom
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

Converting a geometry to an `ogc:geoJSONLiteral` will result in a coordinate transformation to CRS84 Longitude, Latitiude, which is the only coordinate reference system supported by GeoJSON.

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example returns each U.S. Congressional district polygon as an `ogc:geoJSONLiteral`.

```
SELECT name, gjson
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT ?name (ogcf:asGeoJSON(?cgeom) AS ?gjson)
 WHERE
 { ?person usgovt:name ?name .
   ?person pol:hasRole ?role .
   ?role pol:forOffice ?office .
   ?office pol:represents ?cdist .
   ?cdist orageo:hasExactGeometry ?cgeom }'
,sem_models('gov_all_vm'), null
, null ,null, null, ' ALLOW_DUP=T '));
```

B.1.9 ogcf:asGML

Format

ogcf:asGML(geom : geomLiteral, gmlProfile : xsd:string) : ogc:gmlLiteral

Description

Converts `geom` to an equivalent GML representation.

Parameters

geom
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

gmlProfile
This argument is ignored. GML 3.11 profile is used in all cases.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example returns each U.S. Congressional district polygon as an `ogc:GMLLiteral`.

```
SELECT name, gml
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT ?name (ogcf:asGML(?cgeom, "3.11") AS ?gml)
 WHERE
 { ?person usgovt:name ?name .
   ?person pol:hasRole ?role .
   ?role pol:forOffice ?office .
   ?office pol:represents ?cdist .
   ?cdist orageo:hasExactGeometry ?cgeom }'
,sem_models('gov_all_vm'), null
, null ,null, null, ' ALLOW_DUP=T '));
```

B.1.10 ogcf:asKML

Format

ogcf:asKML(geom : geomLiteral) : ogc:kmlLiteral

Description

Converts geom to an equivalent KML representation.

Parameters

geom
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

Converting a geometry to an `ogc:kmlLiteral` will result in a coordinate transformation to CRS84 Longitude, Latitiude, which is the only coordinate reference system supported by GML.

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example returns each U.S. Congressional district polygon as an `ogc:kmlLiteral`.

```
SELECT name, kml
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT ?name (ogcf:asKML(?cgeom) AS ?kml)
 WHERE
 { ?person usgovt:name ?name .
   ?person pol:hasRole ?role .
   ?role pol:forOffice ?office .
   ?office pol:represents ?cdist .
   ?cdist orageo:hasExactGeometry ?cgeom }'
,sem_models('gov_all_vm'), null
, null ,null, null, ' ALLOW_DUP=T '));
```

B.1.11 ogcf:asWKT

Format

ogcf:asWKT(geom : geomLiteral) : ogc:wktLiteral

Description

Converts geom to an equivalent WKT representation.

Parameters

geom
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example returns each U.S. Congressional district polygon as an `ogc:wktLiteral`.

```
SELECT name, wkt
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
```

```
 SELECT ?name (ogcf:asWKT(?cgeom) AS ?wkt)
 WHERE
 { ?person usgovt:name ?name .
   ?person pol:hasRole ?role .
   ?role pol:forOffice ?office .
   ?office pol:represents ?cdist .
   ?cdist orageo:hasExactGeometry ?cgeom }'
,sem_models('gov_all_vm'), null
, null ,null, null, ' ALLOW_DUP=T '));
```

B.1.12 ogcf:boundary

Format

ogcf:boundary(geom : geomLiteral) : ogc:wktLiteral

Description

Returns a geometry object that is the closure of the boundary of `geom`.

Parameters

geom
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example finds the boundaries of U.S. Congressional district polygons.

```
SELECT cb
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT (ogcf:boundary(?cgeom) AS ?cb)
 WHERE
 { ?person usgovt:name ?name .
   ?person pol:hasRole ?role .
   ?role pol:forOffice ?office .
   ?office pol:represents ?cdist .
   ?cdist orageo:hasExactGeometry ?cgeom }'
,sem_models('gov_all_vm'), null
,null
,null, null, ' ALLOW_DUP=T '));
```

B.1.13 ogcf:boundingCircle

Format

ogcf:boundingCircle(geom : geomLiteral) : ogc:wktLiteral

Description

Returns a geometric object that is the minimum bounding circle around `geom`.

Parameters

geom
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example returns the minimum bounding circle around each U.S. Congressional district polygon.

```
SELECT name, bc
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT ?name (ogcf:boundingCircle(?cgeom) AS ?bc)
 WHERE
 { ?person usgovt:name ?name .
   ?person pol:hasRole ?role .
   ?role pol:forOffice ?office .
   ?office pol:represents ?cdist .
   ?cdist orageo:hasExactGeometry ?cgeom }'
,sem_models('gov_all_vm'), null
, null ,null, null, ' ALLOW_DUP=T '));
```

B.1.14 ogcf:buffer

Format

ogcf:buffer(geom : geomLiteral, radius : xsd:decimal, units : xsd:anyURI) : ogc:wktLiteral

Description

Returns a buffer polygon the specified radius (measured in units) around a geometry.

Parameters

geom
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

radius
Radius value used to define the buffer.

units
Unit of measurement:

- A URI of the form `<http://xmlns.oracle.com/rdf/geo/uom/{SDO_UNIT}>` (for example, `<http://xmlns.oracle.com/rdf/geo/uom/M>`). Any SDO_UNIT value from the MDSYS.SDO_DIST_UNITS table will be recognized. See the section about Unit Of Measurement Support in *Oracle Spatial Developer's Guide* for more information about unit of measurement specification.
- A URI from the QUDT vocabulary of units that has an equivalent unit in MDSYS.SDO_DIST_UNITS table. For example, `<http://qudt.org/vocab/unit/M>` for meter.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example finds the U.S. Congressional district polygons that are within a 100–kilometer buffer around a specified point.

```
SELECT name, cdist
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT ?name ?cdist
 WHERE
 { # HINT0={LEADING(?cgeom)}
   ?person usgovt:name ?name .
   ?person pol:hasRole ?role .
   ?role pol:forOffice ?office .
   ?office pol:represents ?cdist .
   ?cdist orageo:hasExactGeometry ?cgeom
   FILTER (
     ogcf:sfWithin(?cgeom,
       ogcf:buffer("POINT(-71.46444 42.7575)"^^ogc:wktLiteral,
                    100,
                    <http://xmlns.oracle.com/rdf/geo/uom/KM>))) }'
,sem_models('gov_all_vm'), null
,null
,null, null, ' ALLOW_DUP=T '));
```

B.1.15 ogcf:concaveHull

Format

ogcf:concaveHull(geom : geomLiteral) : ogc:wktLiteral

Description

Returns a geometric object that represents the concave hull of `geom`. The convex hull is a simple convex polygon that completely encloses the geometry object. Spatial uses as few straight-line sides as possible to create the smallest polygon that completely encloses the specified object. A convex hull is a convenient way to get an approximation of a complex geometry object.

Parameters

geom
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example returns the concave hull of each U.S. Congressional district polygon.

```
SELECT name, ch
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT ?name (ogcf:concaveHull(?cgeom) AS ?ch)
 WHERE
 { ?person usgovt:name ?name .
   ?person pol:hasRole ?role .
   ?role pol:forOffice ?office .
   ?office pol:represents ?cdist .
   ?cdist orageo:hasExactGeometry ?cgeom }'
,sem_models('gov_all_vm'), null
, null ,null, null, ' ALLOW_DUP=T '));
```

B.1.16 ogcf:convexHull

Format

ogcf:convexHull(geom : geomLiteral) : ogc:wktLiteral

Description

Returns a polygon geometry that represents the convex hull of geom. (The convex hull is a simple convex polygon that completely encloses the geometry object, using as few straight-line sides as possible to create the smallest polygon that completely encloses the geometry object.)

Parameters

geom
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example finds the U.S. Congressional district polygons whose convex hull contains a specified point.

```
SELECT name, cdist
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT ?name ?cdist
 WHERE
 { ?person usgovt:name ?name .
   ?person pol:hasRole ?role .
   ?role pol:forOffice ?office .
   ?office pol:represents ?cdist .
   ?cdist orageo:hasExactGeometry ?cgeom
   FILTER (ogcf:sfContains(ogcf:convexHull(?cgeom),
      "POINT(-71.46444 42.7575)"^^ogc:wktLiteral)) } '
,sem_models('gov_all_vm'), null
,null
,null, null, ' ALLOW_DUP=T '));
```

B.1.17 ogcf:coordinateDimension

Format

ogcf:coordinateDimension(geom : geomLiteral) : xsd:integer

Description

Returns the coordinate dimension of `geom`. The coordinate dimension is the number of measurements or axes needed to describe a position in the coordinate reference system of `geom`.

Parameters

geom
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example returns the coordinate dimension of each U.S. Congressional district polygon.

```
SELECT cdist, cd
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT ?cdist (ogcf:coordinateDimension(?cgeom) AS ?cd)
 WHERE
 { ?cdist orageo:hasExactGeometry ?cgeom }'
,sem_models('gov_all_vm'), null
, null ,null, null, ' ALLOW_DUP=T '));
```

B.1.18 ogcf:difference

Format

ogcf:difference(geom1 : geomLiteral, geom2 : geomLiteral) : ogc:wktLiteral

Description

Returns a geometry object that is the topological difference (MINUS operation) of `geom1` and `geom2`.

Parameters

geom1
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

geom2
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example finds the U.S. Congressional district polygons whose centroid is within the difference of two specified polygons.

```
SELECT name, cdist
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT ?name ?cdist
 WHERE
{ ?person usgovt:name ?name .
   ?person pol:hasRole ?role .
   ?role pol:forOffice ?office .
   ?office pol:represents ?cdist .
   ?cdist orageo:hasExactGeometry ?cgeom
   FILTER (ogcf:sfWithin(orageo:centroid(?cgeom),
        ogcf:difference(
          "Polygon((-83.6 34.1, -83.2 34.1, -83.2 34.5, -83.6 34.5, -83.6
34.1))"^^ogc:wktLiteral,
          "Polygon((-83.2 34.3, -83.0 34.3, -83.0 34.5, -83.2 34.5, -83.2
34.3))"^^ogc:wktLiteral))) } '
,sem_models('gov_all_vm'), null
,null, null, null, ' ALLOW_DUP=T '));
```

B.1.19 ogcf:dimension

Format

ogcf:dimension(geom : geomLiteral) : xsd:integer

Description

Returns the dimension of `geom`. For example, the dimension of a point is 0, a line is 1, and a polygon is 2.

.

Parameters

geom
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example returns the dimension of each U.S. Congressional district polygon.

```
SELECT cdist, cd
FROM table(sem_match(
```

```
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT ?cdist (ogcf:dimension(?cgeom) AS ?cd)
 WHERE
 { ?cdist orageo:hasExactGeometry ?cgeom }'
,sem_models('gov_all_vm'), null
, null ,null, null, ' ALLOW_DUP=T '));
```

B.1.20 ogcf:distance

Format

ogcf:distance(geom1 : geomLiteral, geom2 : geomLiteral, units : xsd:anyURI) : xsd:decimal

Description

Returns the distance in units between the two closest points of `geom1` and `geom2`.

Parameters

geom1
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

geom2
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

units
Unit of measurement:

- A URI of the form `<http://xmlns.oracle.com/rdf/geo/uom/{SDO_UNIT}>` (for example, `<http://xmlns.oracle.com/rdf/geo/uom/KM>`). Any SDO_UNIT value from the MDSYS.SDO_DIST_UNITS table will be recognized. See the section about unit of measurement support in *Oracle Spatial and Graph Developer's Guide* for more information about unit of measurement specification.
- A URI from the QUDT vocabulary of units that has an equivalent unit in MDSYS.SDO_DIST_UNITS table. For example, `<http://qudt.org/vocab/unit/M>` for meter.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example orders U.S. Congressional districts based on distance from a specified point.

```
SELECT name, cdist
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
```

```
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT ?name ?cdist
 WHERE
 { # HINT0={LEADING(?cgeom)}
   ?person usgovt:name ?name .
   ?person pol:hasRole ?role .
   ?role pol:forOffice ?office .
   ?office pol:represents ?cdist .
   ?cdist orageo:hasExactGeometry ?cgeom
 }
 ORDER BY ASC(ogcf:distance(?cgeom,
                "POINT(-71.46444 42.7575)"^^ogc:wktLiteral,
                <http://xmlns.oracle.com/rdf/geo/uom/KM>))'
,sem_models('gov_all_vm'), null
,null, null, null, ' ALLOW_DUP=T '))
ORDER BY sem$rownum;
```

B.1.21 ogcf:envelope

Format

ogcf:envelope(geom : geomLiteral) : ogc:wktLiteral

Description

Returns the minimum bounding rectangle (MBR) of `geom`, that is, the single rectangle that minimally encloses `geom`.

Parameters

geom
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example finds the U.S. Congressional district polygons whose minimum bounding rectangle contains a specified point.

```
SELECT name, cdist
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT ?name ?cdist
```

```
 WHERE
 { ?person usgovt:name ?name .
   ?person pol:hasRole ?role .
   ?role pol:forOffice ?office .
   ?office pol:represents ?cdist .
   ?cdist orageo:hasExactGeometry ?cgeom
   FILTER (ogcf:sfContains(ogcf:envelope(?cgeom),
      "POINT(-71.46444 42.7575)"^^ogc:wktLiteral)) } '
,sem_models('gov_all_vm'), null
,null, null, null, ' ALLOW_DUP=T '));
```

B.1.22 ogcf:geometryN

Format

ogcf:geometryN(geom : geomLiteral, geomindex : xsd:integer) : ogc:wktLiteral

Description

Returns the n^{th} geometry of `geom` if `geom` is a geometry collection or `geom` if `geom` is a single geometry and `n=1`.

Parameters

geom
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

geomindex
The position of the desired geometry in the collection. The first geometry has an index of 1.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example returns the second geometry in the input geometry collection.

```
SELECT g
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT (ogcf:geometryN("GEOMETRYCOLLECTION(POINT(-75 44), LINESTRING(-75 44,
-75 45),
         POLYGON((-75 44, -75 43, -74 43, -74 44, -75 44)))"^^ogc:wktLiteral,
2) AS ?g)
 WHERE
 { }'
,sem_models('gov_all_vm'), null
, null ,null, null, ' ALLOW_DUP=T '));
```

B.1.23 ogcf:geometryType

Format

ogcf:geometryType(geom : geomLiteral) : xsd:anyURI

Description

Returns the URI of the subtype of `<http://www.opengis.net/ont/sf#Geometry>` of which `geom` is a member. Possible return values are:

- `<http://www.opengis.net/ont/sf#Point>`
- `<http://www.opengis.net/ont/sf#LineString>`
- `<http://www.opengis.net/ont/sf#Polygon>`
- `<http://www.opengis.net/ont/sf#GeometryCollection>`
- `<http://www.opengis.net/ont/sf#MultiPoint>`
- `<http://www.opengis.net/ont/sf#MultiLineString>`
- `<http://www.opengis.net/ont/sf#MultiPolygon>`
- `<http://www.opengis.net/ont/sf#Solid>`
- `<http://www.opengis.net/ont/sf#MultiSolid>`
- `<http://www.opengis.net/ont/sf#Unknown>`

Parameters

geom
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example returns the geometry type of each U.S. Congressional district polygon.

```
SELECT cdist, gtype
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT ?cdist (ogcf:geometryType(?cgeom) AS ?gtype)
 WHERE
 { ?cdist orageo:hasExactGeometry ?cgeom }'
,sem_models('gov_all_vm'), null
, null ,null, null, ' ALLOW_DUP=T '));
```

B.1.24 ogcf:getSRID

Format

ogcf:getSRID(geom : geomLiteral) : xsd:anyURI

Description

Returns the spatial reference system URI for `geom`.

Parameters

geom
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

The URI returned has the form `<http://www.opengis.net/def/crs/EPSG/0/{srid}>`, where `{srid}` is a valid spatial reference system ID from the European Petroleum Survey Group (EPSG).

For URIs that are not in the EPSG Geodetic Parameter Dataset, the URI returned has the form `<http://xmlns.oracle.com/rdf/geo/srid/{srid}>` , where `{srid}` is a valid spatial reference system ID from Oracle Spatial and Graph.

For the default spatial reference system, WGS84 Longitude-Latitude, the URI `<http://www.opengis.net/def/crs/OGC/1.3/CRS84>` is returned.

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example finds spatial reference system URIs for U.S. Congressional district polygons.

```
SELECT csrid
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT (ogcf:getSRID(?cgeom) AS ?csrid)
 WHERE
 { ?person usgovt:name ?name .
   ?person pol:hasRole ?role .
   ?role pol:forOffice ?office .
   ?office pol:represents ?cdist .
   ?cdist orageo:hasExactGeometry ?cgeom }'
,sem_models('gov_all_vm'), null
,null, null, null, ' ALLOW_DUP=T '));
```

B.1.25 ogcf:intersection

Format

ogcf:intersection (geom1 : geomLiteral, geom2 : geomLiteral) : ogc:wktLiteral

Description

Returns a geometry object that is the topological intersection (AND operation) of `geom1` and `geom2`.

Parameters

geom1
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

geom2
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example finds the U.S. Congressional district polygons whose centroid is within the intersection of two specified polygons.

```
SELECT name, cdist
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT ?name ?cdist
 WHERE
 { ?person usgovt:name ?name .
   ?person pol:hasRole ?role .
   ?role pol:forOffice ?office .
   ?office pol:represents ?cdist .
   ?cdist orageo:hasExactGeometry ?cgeom
   FILTER (ogcf:sfWithin(orageo:centroid(?cgeom),
        ogcf:intersection(
           "Polygon((-83.6 34.1, -83.2 34.1, -83.2 34.5, -83.6 34.5, -83.6 
34.1))"^^ogc:wktLiteral,
           "Polygon((-83.2 34.3, -83.0 34.3, -83.0 34.5, -83.2 34.5, -83.2 
34.3))"^^ogc:wktLiteral))) } '
,sem_models('gov_all_vm'), null
,null, null, null, ' ALLOW_DUP=T '));
```

B.1.26 ogcf:is3D

Format

ogcf:is3D(geom : geomLiteral) : xsd:boolean

Description

Returns `true` if the spatial dimension of geom is 3.

Parameters

geom
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example checks to see if there are any 3-dimensional U.S. Congressional district polygons.

```
SELECT ask
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 ASK
 WHERE
 { ?cdist orageo:hasExactGeometry ?cgeom
   FILTER(ogcf:is3D(?cgeom)) }'
,sem_models('gov_all_vm'), null
, null ,null, null, ' ALLOW_DUP=T '));
```

B.1.27 ogcf:isEmpty

Format

ogcf:isEmpty(geom : geomLiteral) : xsd:boolean

Description

Returns `true` if `geom` is an empty geometry.

Parameters

geom
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example checks to see if there are any empty U.S. Congressional district geometries.

```
SELECT ask
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 ASK
 WHERE
 { ?cdist orageo:hasExactGeometry ?cgeom
   FILTER(ogcf:isEmpty(?cgeom)) }'
,sem_models('gov_all_vm'), null
, null ,null, null, ' ALLOW_DUP=T '));
```

B.1.28 ogcf:isMeasured

Format

ogcf:isMeasured(geom : geomLiteral) : xsd:boolean

Description

Returns `true` if `geom` has a measure value.

Parameters

geom
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example checks to see if there are any U.S. Congressional district geometries with measure values.

```
SELECT ask
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
```

```
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 ASK
 WHERE
 { ?cdist orageo:hasExactGeometry ?cgeom
   FILTER(ogcf:isMeasured(?cgeom)) }'
,sem_models('gov_all_vm'), null
, null ,null, null, ' ALLOW_DUP=T '));
```

B.1.29 ogcf:isSimple

Format

ogcf:isSimple(geom : geomLiteral) : xsd:boolean

Description

Returns `true` if `geom` is a simple geometry. That is, the geometry has no inconsistent features such as self intersection, identical consecutive vertices, and so on.

Parameters

geom
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT is used to determine if a geometry is simple. The geometry is simple if SDO_GEOM.VALIDATE_GEOMETRY_WITH_CONTEXT returns `TRUE`.

See also the OGC GeoSPARQL specification.

Example

The following example returns any non-simple Congressional district geometries.

```
SELECT cdist, cgeom
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT ?cdist ?cgeom
 WHERE
 { ?cdist orageo:hasExactGeometry ?cgeom
   FILTER(!ogcf:isSimple(?cgeom)) }'
,sem_models('gov_all_vm'), null
, null ,null, null, ' ALLOW_DUP=T '));
```

B.1.30 ogcf:length

Format

ogcf:length(geom : geomLiteral, units : xsd:anyURI) : xsd:double

Description

Returns the length of the `geom`. The length is the maximum distance between any two points in the `geom`.

Parameters

geom
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

units
Unit of measurement:

- A URI of the form `<http://xmlns.oracle.com/rdf/geo/uom/{SDO_UNIT}>` (for example, `<http://xmlns.oracle.com/rdf/geo/uom/M>`). Any SDO_UNIT value from the MDSYS.SDO_DISTANCE_UNITS table will be recognized. See the section about Unit Of Measurement Support in *Oracle Spatial Developer's Guide* for more information about unit of measurement specification.
- A URI from the QUDT vocabulary of units that has an equivalent unit in MDSYS.SDO_DISTANCE_UNITS table. For example, `<http://qudt.org/vocab/unit/M>` for meter.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example finds the maximum length in meters of the longest U.S. Congressional district.

```
SELECT maxl
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT (max(ogcf:length(?cgeom, <http://qudt.org/vocab/unit/M>)) AS ?maxl)
 WHERE
 { ?cdist orageo:hasExactGeometry ?cgeom }'
,sem_models('gov_all_vm'), null
, null ,null, null, ' ALLOW_DUP=T '));
```

B.1.31 ogcf:maxX

Format

ogcf:maxX(geom : geomLiteral) : xsd:double

Description

Returns the maximum X coordinate value for `geom`.

Parameters

geom
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example finds the maximum X coordinate value for each U.S. Congressional district.

```
SELECT cdist, maxX
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT ?cdist (ogcf:maxX(?cgeom) AS ?maxX)
 WHERE
 { ?cdist orageo:hasExactGeometry ?cgeom }'
,sem_models('gov_all_vm'), null
, null ,null, null, ' ALLOW_DUP=T '));
```

B.1.32 ogcf:maxY

Format

ogcf:maxY(geom : geomLiteral) : xsd:double

Description

Returns the maximum Y coordinate value for `geom`.

Parameters

geom
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example finds the maximum Y coordinate value for each U.S. Congressional district.

```
SELECT cdist, maxY
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT ?cdist (ogcf:maxY(?cgeom) AS ?maxY)
 WHERE
 { ?cdist orageo:hasExactGeometry ?cgeom }'
,sem_models('gov_all_vm'), null
, null ,null, null, ' ALLOW_DUP=T '));
```

B.1.33 ogcf:maxZ

Format

ogcf:maxZ(geom : geomLiteral) : xsd:double

Description

Returns the maximum Z coordinate value for `geom`.

Parameters

geom
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example finds the maximum Z coordinate value for a constant geometry.

```
SELECT maxZ
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
```

```
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT (ogcf:maxZ("<http://www.opengis.net/def/crs/EPSG/0/4327> POLYGON((-75
44 10, -75 43 11,
             -74 43 11, -74 44 11, -75 44 10))"^^ogc:wktLiteral) AS ?maxZ)
 WHERE
 {  }'
,sem_models('gov_all_vm'), null
, null ,null, null, ' ALLOW_DUP=T '));
```

B.1.34 ogcf:metricArea

Format

ogcf:metricArea(geom : geomLiteral) : xsd:double

Description

Returns the area of `geom` in square meters.

Parameters

geom
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example finds the area in square meters for each U.S. Congressional district.

```
SELECT name, ma
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT ?name (ogcf:metricArea(?cgeom) AS ?ma)
 WHERE
 { ?person usgovt:name ?name .
   ?person pol:hasRole ?role .
   ?role pol:forOffice ?office .
   ?office pol:represents ?cdist .
   ?cdist orageo:hasExactGeometry ?cgeom }'
,sem_models('gov_all_vm'), null
, null ,null, null, ' ALLOW_DUP=T '));
```

B.1.35 ogcf:metricBuffer

Format

ogcf:metricBuffer(geom : geomLiteral, radius : xsd:double) : ogc:wtkLiteral

Description

Returns a buffer polygon with the specified radius in meters around a geometry.

Parameters

geom
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

radius
Radius value in meters used to define the buffer.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example finds the U.S. Congressional district polygons that are within a 1000–meter buffer around a specified point.

```
SELECT name, cdist
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT ?name ?cdist
 WHERE
 { # HINT0={LEADING(?cgeom)}
   ?person usgovt:name ?name .
   ?person pol:hasRole ?role .
   ?role pol:forOffice ?office .
   ?office pol:represents ?cdist .
   ?cdist orageo:hasExactGeometry ?cgeom
   FILTER (
     ogcf:sfWithin(?cgeom,
       ogcf:metricBuffer("POINT(-71.46444 42.7575)"^^ogc:wktLiteral,
                         1000)))
 }'
,sem_models('gov_all_vm'), null
,null
,null, null, ' ALLOW_DUP=T '));
```

B.1.36 ogcf:metricLength

Format

ogcf:metricLength(geom : geomLiteral) : xsd:double

Description

Returns the length of `geom` in meters. The length is the maximum distance between any two points in `geom`.

Parameters

geom
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example finds the maximum length in meters of the longest U.S. Congressional district.

```
SELECT cdist, maxl
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT ?cdist (max(ogcf:metricLength(?cgeom)) AS ?maxl)
 WHERE
 { ?cdist orageo:hasExactGeometry ?cgeom }'
,sem_models('gov_all_vm'), null
, null ,null, null, ' ALLOW_DUP=T '));
```

B.1.37 ogcf:metricPerimeter

Format

ogcf:metricPerimeter(geom : geomLiteral) : xsd:double

Description

Returns the length of the outer boundary of `geom` in meters.

Parameters

geom
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example finds the maximum perimeter in meters across the set of U.S. Congressional districts.

```
SELECT cdist, maxp
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT ?cdist (max(ogcf:metricPerimeter(?cgeom)) AS ?maxp)
 WHERE
 { ?cdist orageo:hasExactGeometry ?cgeom }'
,sem_models('gov_all_vm'), null
, null ,null, null, ' ALLOW_DUP=T '));
```

B.1.38 ogcf:minX

Format

ogcf:minX(geom : geomLiteral) : xsd:double

Description

Returns the minimum X coordinate value for `geom`.

Parameters

geom
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example finds the minimum X coordinate value for each U.S. Congressional district.

```
SELECT cdist, minX
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT ?cdist (ogcf:minX(?cgeom) AS ?minX)
 WHERE
 { ?cdist orageo:hasExactGeometry ?cgeom }'
,sem_models('gov_all_vm'), null
, null ,null, null, ' ALLOW_DUP=T '));
```

B.1.39 ogcf:minY

Format

ogcf:minY(geom : geomLiteral) : xsd:double

Description

Returns the minimum Y coordinate value for `geom`.

Parameters

geom
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example finds the minimum Y coordinate value for each U.S. Congressional district.

```
SELECT cdist, minY
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT ?cdist (ogcf:minY(?cgeom) AS ?minY)
 WHERE
 { ?cdist orageo:hasExactGeometry ?cgeom }'
```

```
,sem_models('gov_all_vm'), null
, null ,null, null, ' ALLOW_DUP=T '));
```

B.1.40 ogcf:minZ

Format

ogcf:minZ(geom : geomLiteral) : xsd:double

Description

Returns the minimum Z coordinate value for `geom`.

Parameters

geom
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example finds the minimum Z coordinate value for a constant geometry.

```
SELECT minZ
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT (ogcf:minZ("<http://www.opengis.net/def/crs/EPSG/0/4327> POLYGON((-75
44 10, -75 43 11, -74 43 11,
                         -74 44 11, -75 44 10))"^^ogc:wktLiteral) AS ?minZ)
 WHERE
 {  }'
,sem_models('gov_all_vm'), null
, null ,null, null, ' ALLOW_DUP=T '));
```

B.1.41 ogcf:numGeometries

Format

ogcf:numGeometries(geom : geomLiteral) : xsd:int

Description

Returns the number of geometries in `geom`.

Parameters

geom
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example finds the number of geometries in a constant geometry collection.

```
SELECT ng
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT (ogcf:numGeometries("GEOMETRYCOLLECTION(POINT(-75 44), LINESTRING(-75
44, -75 45), POLYGON((-75 44, -75 43, -74 43, -74 44, -75
44)))"^^ogc:wktLiteral) AS ?ng)
 WHERE
 {  }'
,sem_models('gov_all_vm'), null
, null ,null, null, ' ALLOW_DUP=T '));
```

B.1.42 ogcf:perimeter

Format

ogcf:perimeter(geom : geomLiteral, units : xsd:anyURI) : xsd:double

Description

Returns the length of the outer boundary of `geom` measured in `units`.

Parameters

geom
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

units
Unit of measurement:

- A URI of the form `<http://xmlns.oracle.com/rdf/geo/uom/{SDO_UNIT}>` (for example, `<http://xmlns.oracle.com/rdf/geo/uom/M>`). Any SDO_UNIT value from the MDSYS.SDO_DISTANCE_UNITS table will be recognized. See the section about Unit Of Measurement Support in *Oracle Spatial Developer's Guide* for more information about unit of measurement specification.

- A URI from the QUDT vocabulary of units that has an equivalent unit in MDSYS.SDO_DISTANCE_UNITS table. For example, `<http://qudt.org/vocab/unit/M>` for meter.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example finds the maximum perimeter in meters across the set of U.S. Congressional districts.

```
SELECT maxp
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT (max(ogcf:perimeter(?cgeom, <http://qudt.org/vocab/unit/M>)) AS ?maxp)
 WHERE
 { ?cdist orageo:hasExactGeometry ?cgeom }'
,sem_models('gov_all_vm'), null
, null ,null, null, ' ALLOW_DUP=T '));
```

B.1.43 ogcf:relate

Format

ogcf:relate(geom1 : geomLiteral, geom2 : geomLiteral, pattern-matrix : xsd:string) : xsd:boolean

Description

Returns `true` if the topological relationship between `geom1` and `geom2` satisfies the specified DE-9IM pattern-matrix. Returns `false` otherwise.

Parameters

geom1
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

geom2
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

pattern-matrix
A dimensionally extended 9-intersection model (DE-9IM) intersection pattern string consisting of `T` (true) and `F` (false) values. A DE-9IM pattern string describes the intersections between the interiors, boundaries, and exteriors of two geometries.

Usage Notes

When invoking `ogcf:relate` with a query variable and a constant geometry, always use the query variable as the first parameter and the constant geometry as the second parameter.

For best performance, `geom1` should be a local variable (that is, a variable that appears in the basic graph pattern that contains the `ogcf:relate` spatial filter).

It is recommended to use a `LEADING(?var)` HINT0 hint when the query involves a restrictive `ogcf:relate` spatial filter on `?var`.

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See the OGC Simple Features Specification (OGC 06-103r3) for a detailed description of DE-9IM intersection patterns. See also the OGC GeoSPARQL specification.

Example

The following example finds the U.S. Congressional district that contains a specified point.

```
SELECT name, cdist
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT ?name ?cdist
 WHERE
 { # HINT0={LEADING(?cgeom)}
   ?person usgovt:name ?name .
   ?person pol:hasRole ?role .
   ?role pol:forOffice ?office .
   ?office pol:represents ?cdist .
   ?cdist orageo:hasExactGeometry ?cgeom
   FILTER (ogcf:relate(?cgeom,
      "POINT(-71.46444 42.7575)"^^ogc:wktLiteral,
      "TTTFFTFFT")) } '
,sem_models('gov_all_vm'), null
,null, null, null, ' ALLOW_DUP=T '
));
```

B.1.44 ogcf:sfContains

Format

ogcf:sfContains(geom1 : geomLiteral, geom2 : geomLiteral) : xsd:boolean

Description

Returns `true` if `geom1` spatially contains `geom2` as defined by the OGC Simple Features specification (OGC 06-103r3). Returns `false` otherwise.

Parameters

geom1
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

geom2
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

When invoking this function with a query variable and a constant geometry, always use the query variable as the first parameter and the constant geometry as the second parameter.

For best performance, `geom1` should be a local variable (that is, a variable that appears in the basic graph pattern that contains the `ogcf:sfContains` spatial filter).

It is recommended to use a `LEADING(?var)` HINT0 hint when the query involves a restrictive `ogcf:sfContains` spatial filter on `?var`.

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example finds U.S. Congressional district polygons that spatially contain a constant polygon.

```
SELECT name, cdist
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT ?name ?cdist
 WHERE
 { # HINT0={LEADING(?cgeom)}
   ?person usgovt:name ?name .
   ?person pol:hasRole ?role .
   ?role pol:forOffice ?office .
   ?office pol:represents ?cdist .
   ?cdist orageo:hasExactGeometry ?cgeom
   FILTER (ogcf:sfContains(?cgeom,
             "Polygon((-83.6 34.1, -83.2 34.1, -83.2 34.5, -83.6 34.5, -83.6
34.1))"^^ogc:wktLiteral)) } '
,sem_models('gov_all_vm'), null
,null, null, null, ' ALLOW_DUP=T '));
```

B.1.45 ogcf:sfCrosses

Format

ogcf:sfCrosses(geom1 : geomLiteral, geom2 : geomLiteral) : xsd:boolean

Description

Returns `true` if `geom1` spatially crosses `geom2` as defined by the OGC Simple Features specification (OGC 06-103r3). Returns `false` otherwise.

Parameters

geom1
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

geom2
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

When invoking this function with a query variable and a constant geometry, always use the query variable as the first parameter and the constant geometry as the second parameter.

For best performance, `geom1` should be a local variable (that is, a variable that appears in the basic graph pattern that contains the `ogcf:sfCrosses` spatial filter).

It is recommended to use a `LEADING(?var)` HINT0 hint when the query involves a restrictive `ogcf:sfCrosses` spatial filter on `?var`.

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example finds U.S. Congressional district polygons that spatially cross a constant polygon.

```
SELECT name, cdist
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT ?name ?cdist
 WHERE
 { # HINT0={LEADING(?cgeom)}
   ?person usgovt:name ?name .
   ?person pol:hasRole ?role .
   ?role pol:forOffice ?office .
   ?office pol:represents ?cdist .
   ?cdist orageo:hasExactGeometry ?cgeom
   FILTER (ogcf:sfCrosses(?cgeom,
             "Polygon((-83.6 34.1, -83.2 34.1, -83.2 34.5, -83.6 34.5, -83.6
34.1))"^^ogc:wktLiteral)) } '
,sem_models('gov_all_vm'), null
,null, null, null, ' ALLOW_DUP=T '));
```

B.1.46 ogcf:sfDisjoint

Format

ogcf:fDisjoint(geom1 : geomLiteral, geom2 : geomLiteral) : xsd:boolean

Description

Returns `true` if the two geometries are spatially disjoint as defined by the OGC Simple Features specification (OGC 06-103r3). Returns `false` otherwise.

Parameters

geom1
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

geom2
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

The `ogcf:sfDisjoint` filter cannot use a spatial index for evaluation, so performance will probably be much worse than with other simple features spatial functions.

See also the OGC GeoSPARQL specification.

Example

The following example finds U.S. Congressional district polygons that are spatially disjoint from a constant polygon.

```
SELECT name, cdist
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT ?name ?cdist
 WHERE
 { # HINT0={LEADING(?cgeom)}
   ?person usgovt:name ?name .
   ?person pol:hasRole ?role .
   ?role pol:forOffice ?office .
   ?office pol:represents ?cdist .
   ?cdist orageo:hasExactGeometry ?cgeom
   FILTER (ogcf:sfDisjoint(?cgeom,
             "Polygon((-83.6 34.1, -83.2 34.1, -83.2 34.5, -83.6 34.5, -83.6
34.1))"^^ogc:wktLiteral)) } '
,sem_models('gov_all_vm'), null
,null, null, null, ' ALLOW_DUP=T '));
```

B.1.47 ogcf:sfEquals

Format

ogcf:sfEquals(geom1 : geomLiteral, geom2 : geomLiteral) : xsd:boolean

Description

Returns `true` if the two geometries are spatially equal as defined by the OGC Simple Features specification (OGC 06-103r3). Returns `false` otherwise.

Parameters

geom1
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

geom2
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

When invoking this function with a query variable and a constant geometry, always use the query variable as the first parameter and the constant geometry as the second parameter.

For best performance, `geom1` should be a local variable (that is, a variable that appears in the basic graph pattern that contains the `ogcf:sfEquals` spatial filter).

It is recommended to use a `LEADING(?var)` HINT0 hint when the query involves a restrictive `ogcf:sfEquals` spatial filter on `?var`.

See Spatial Support for information about representing , indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example finds U.S. Congressional district polygons that are spatially equal to a constant polygon.

```
SELECT name, cdist
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT ?name ?cdist
 WHERE
 { # HINT0={LEADING(?cgeom)}
   ?person usgovt:name ?name .
   ?person pol:hasRole ?role .
   ?role pol:forOffice ?office .
   ?office pol:represents ?cdist .
   ?cdist orageo:hasExactGeometry ?cgeom
   FILTER (ogcf:sfEquals(?cgeom,
             "Polygon((-83.6 34.1, -83.2 34.1, -83.2 34.5, -83.6 34.5, -83.6
```

```
34.1))"^^ogc:wktLiteral)) } '
,sem_models('gov_all_vm'), null
,null, null, null, ' ALLOW_DUP=T '));
```

B.1.48 ogcf:sfIntersects

Format

ogcf:sfIntersects(geom1 : geomLiteral, geom2 : geomLiteral) : xsd:boolean

Description

Returns `true` if the two geometries are *not* disjoint as defined by the OGC Simple Features specification (OGC 06-103r3). Returns `false` otherwise.

Parameters

geom1
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

geom2
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

When invoking this function with a query variable and a constant geometry, always use the query variable as the first parameter and the constant geometry as the second parameter.

For best performance, `geom1` should be a local variable (that is, a variable that appears in the basic graph pattern that contains the `ogcf:sfIntersects` spatial filter).

It is recommended to use a `LEADING(?var)` HINT0 hint when the query involves a restrictive `ogcf:sfIntersects` spatial filter on `?var`.

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example finds U.S. Congressional district polygons that intersect a constant polygon.

```
SELECT name, cdist
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT ?name ?cdist
 WHERE
{  # HINT0={LEADING(?cgeom)}
   ?person usgovt:name ?name .
   ?person pol:hasRole ?role .
   ?role pol:forOffice ?office .
```

```
   ?office pol:represents ?cdist .
   ?cdist orageo:hasExactGeometry ?cgeom
   FILTER (ogcf:sfIntersects(?cgeom,
             "Polygon((-83.6 34.1, -83.2 34.1, -83.2 34.5, -83.6 34.5, -83.6
34.1))"^^ogc:wktLiteral)) } '
,sem_models('gov_all_vm'), null
,null, null, null, ' ALLOW_DUP=T '));
```

B.1.49 ogcf:sfOverlaps

Format

ogcf:sfOverlaps(geom1 : geomLiteral, geom2 : geomLiteral) : xsd:boolean

Description

Returns `true` if `geom1` spatially overlaps `geom2` as defined by the OGC Simple Features specification (OGC 06-103r3). Returns `false` otherwise.

Parameters

geom1
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

geom2
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

When invoking this function with a query variable and a constant geometry, always use the query variable as the first parameter and the constant geometry as the second parameter.

For best performance, `geom1` should be a local variable (that is, a variable that appears in the basic graph pattern that contains the `ogcf:sfOverlaps` spatial filter).

It is recommended to use a `LEADING(?var)` HINT0 hint when the query involves a restrictive `ogcf:sfOverlaps` spatial filter on `?var`.

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example finds U.S. Congressional district polygons that spatially overlap a constant polygon.

```
SELECT name, cdist
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT ?name ?cdist
 WHERE
```

```
{ # HINT0={LEADING(?cgeom)}
   ?person usgovt:name ?name .
   ?person pol:hasRole ?role .
   ?role pol:forOffice ?office .
   ?office pol:represents ?cdist .
   ?cdist orageo:hasExactGeometry ?cgeom
   FILTER (ogcf:sfOverlaps(?cgeom,
             "Polygon((-83.6 34.1, -83.2 34.1, -83.2 34.5, -83.6 34.5, -83.6
34.1))"^^ogc:wktLiteral)) } '
,sem_models('gov_all_vm'), null
,null, null, null, ' ALLOW_DUP=T '));
```

B.1.50 ogcf:sfTouches

Format

ogcf:sfTouches(geom1 : geomLiteral, geom2 : geomLiteral) : xsd:boolean

Description

Returns `true` if the two geometries spatially touch as defined by the OGC Simple Features specification (OGC 06-103r3). Returns `false` otherwise.

Parameters

geom1
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

geom2
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

When invoking this function with a query variable and a constant geometry, always use the query variable as the first parameter and the constant geometry as the second parameter.

For best performance, `geom1` should be a local variable (that is, a variable that appears in the basic graph pattern that contains the `ogcf:sfTouches` spatial filter).

It is recommended to use a `LEADING(?var)` HINT0 hint when the query involves a restrictive `ogcf:sfTouches` spatial filter on `?var`.

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example finds U.S. Congressional district polygons that spatially touch a constant polygon.

```
SELECT name, cdist
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
```

```
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT ?name ?cdist
 WHERE
 { # HINT0={LEADING(?cgeom)}
   ?person usgovt:name ?name .
   ?person pol:hasRole ?role .
   ?role pol:forOffice ?office .
   ?office pol:represents ?cdist .
   ?cdist orageo:hasExactGeometry ?cgeom
   FILTER (ogcf:sfTouches(?cgeom,
             "Polygon((-83.6 34.1, -83.2 34.1, -83.2 34.5, -83.6 34.5, -83.6
34.1))"^^ogc:wktLiteral)) } '
,sem_models('gov_all_vm'), null
,null, null, null, ' ALLOW_DUP=T '));
```

B.1.51 ogcf:sfWithin

Format

ogcf:sfWithin(geom1 : geomLiteral, geom2 : geomLiteral) : xsd:boolean

Description

Returns `true` if `geom1` is spatially within `geom2` as defined by the OGC Simple Features specification (OGC 06-103r3). Returns `false` otherwise.

Parameters

geom1
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

geom2
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

When invoking this function with a query variable and a constant geometry, always use the query variable as the first parameter and the constant geometry as the second parameter.

For best performance, `geom1` should be a local variable (that is, a variable that appears in the basic graph pattern that contains the `ogcf:sfWithin` spatial filter).

It is recommended to use a `LEADING(?var)` HINT0 hint when the query involves a restrictive `ogcf:sfWithin` spatial filter on `?var`.

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example finds U.S. Congressional district polygons that are spatially within a constant polygon.

```
SELECT name, cdist
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT ?name ?cdist
 WHERE
 { # HINT0={LEADING(?cgeom)}
   ?person usgovt:name ?name .
   ?person pol:hasRole ?role .
   ?role pol:forOffice ?office .
   ?office pol:represents ?cdist .
   ?cdist orageo:hasExactGeometry ?cgeom
   FILTER (ogcf:sfWithin(?cgeom,
             "Polygon((-83.6 34.1, -83.2 34.1, -83.2 34.5, -83.6 34.5, -83.6
34.1))"^^ogc:wktLiteral)) } '
,sem_models('gov_all_vm'), null
,null, null, null, ' ALLOW_DUP=T '));
```

B.1.52 ogcf:spatialDimension

Format

ogcf:spatialDimension(geom : geomLiteral) : xsd:integer

Description

Returns the spatial dimension of `geom`. That is, the number of dimensions used for the spatial coordinates of `geom`. It does not include any dimensions used for measure values.

Parameters

geom
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example returns the spatial dimension of each U.S. Congressional district polygon.

```
SELECT cdist, sd
FROM table(sem_match(
```

```
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT ?cdist (ogcf:spatialDimension(?cgeom) AS ?sd)
 WHERE
 { ?cdist orageo:hasExactGeometry ?cgeom }'
,sem_models('gov_all_vm'), null
, null ,null, null, ' ALLOW_DUP=T '));
```

B.1.53 ogcf:symDifference

Format

ogcf:symDifference(geom1 : geomLiteral, geom2 : geomLiteral) : ogc:wktLiteral

Description

Returns a geometry object that is the topological symmetric difference (XOR operation) of `geom1` and `geom2`.

Parameters

geom1
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

geom2
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example finds the U.S. Congressional district polygons that are within a 100-kilometer buffer around a specified point.

```
SELECT name, cdist
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT ?name ?cdist
 WHERE
 { ?person usgovt:name ?name .
   ?person pol:hasRole ?role .
   ?role pol:forOffice ?office .
   ?office pol:represents ?cdist .
   ?cdist orageo:hasExactGeometry ?cgeom
   FILTER (ogcf:sfWithin(orageo:centroid(?cgeom),
```

```
                ogcf:symDifference(
                   "Polygon((-83.6 34.1, -83.2 34.1, -83.2 34.5, -83.6 34.5,
-83.6 34.1))"^^ogc:wktLiteral,
                   "Polygon((-83.2 34.3, -83.0 34.3, -83.0 34.5, -83.2 34.5,
-83.2 34.3))"^^ogc:wktLiteral))) } '
,sem_models('gov_all_vm'), null
,null, null, null, ' ALLOW_DUP=T '));
```

B.1.54 ogcf:transform

Format

ogcf:transform(geom : geomLiteral, srsIRI xsd:anyURI) : ogc:wktLiteral

Description

Transforms `geom` to the spatial reference system defined by srsIRI.

Parameters

geom
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

srsIRI
The target spatial reference system IRI.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Supported spatial reference system IRIs have the following form `<http://www.opengis.net/def/crs/EPSG/0/{srid}>`, where `{srid}` is a valid spatial reference system ID defined by the European Petroleum Survey Group (EPSG). For IRIs that are not in the EPSG Geodetic Parameter Dataset, spatial reference system IRIs of the following form are supported `<http://xmlns.oracle.com/rdf/geo/srid/{srid}>`, where `{srid}` is a valid spatial reference system ID from Oracle Spatial.

Example

The following example projects each Congressional distinct polygon to the NH state plane coordinate reference system (EPSG:3613).

```
SELECT cdist, nhg
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT ?cdist (ogcf:transform(?cgeom, <http://www.opengis.net/def/crs/
EPSG/0/3613>) AS ?nhg)
 WHERE
 { ?cdist orageo:hasExactGeometry ?cgeom }'
```

```
,sem_models('gov_all_vm'), null
, null ,null, null, ' ALLOW_DUP=T '));
```

B.1.55 ogcf:union

Format

ogcf:union(geom1 : geomLiteral, geom2 : geomLiteral) : ogc:wktLiteral

Description

Returns a geometry object that is the topological union (OR operation) of `geom1` and `geom2`.

Parameters

geom1
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

geom2
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the OGC GeoSPARQL specification.

Example

The following example finds the U.S. Congressional district polygons whose centroid is within the union of two specified polygons.

```
SELECT name, cdist
FROM table(sem_match(
'PREFIX ogc: <http://www.opengis.net/ont/geosparql#>
 PREFIX ogcf: <http://www.opengis.net/def/function/geosparql/>
 PREFIX orageo: <http://xmlns.oracle.com/rdf/geo/>
 PREFIX pol: <http://www.rdfabout.com/rdf/schema/politico/>
 PREFIX usgovt: <http://www.rdfabout.com/rdf/schema/usgovt/>
 SELECT ?name ?cdist
 WHERE
 { ?person usgovt:name ?name .
   ?person pol:hasRole ?role .
   ?role pol:forOffice ?office .
   ?office pol:represents ?cdist .
   ?cdist orageo:hasExactGeometry ?cgeom
   FILTER (ogcf:sfWithin(orageo:centroid(?cgeom),
        ogcf:union(
          "Polygon((-83.6 34.1, -83.2 34.1, -83.2 34.5, -83.6 34.5, -83.6
34.1))"^^ogc:wktLiteral,
          "Polygon((-83.2 34.3, -83.0 34.3, -83.0 34.5, -83.2 34.5, -83.2
34.3))"^^ogc:wktLiteral))) } '
,sem_models('gov_all_vm'), null
,null, null, null, ' ALLOW_DUP=T '));
```

B.2 Oracle-Specific Functions for Spatial Support

This section provides reference information about the Oracle-specific functions:

- orageo:aggrCentroid
- orageo:aggrConvexHull
- orageo:aggrMBR
- orageo:aggrUnion
- orageo:area
- orageo:buffer
- orageo:centroid
- orageo:convexHull
- orageo:difference
- orageo:distance
- orageo:getSRID
- orageo:intersection
- orageo:length
- orageo:mbr
- orageo:nearestNeighbor
- orageo:relate
- orageo:sdoDistJoin
- orageo:sdoJoin
- orageo:union
- orageo:withinDistance
- orageo:xor

B.2.1 orageo:aggrCentroid

Format

orageo:aggrCentroid(geom : geomLiteral) : ogc:wktLiteral

Description

Returns a geometry literal that is the centroid of the group of specified geometry objects. (The centroid is also known as the "center of gravity.")

Parameters

geom
Geometry objects. Specified as a query variable.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the SDO_AGGR_CENTROID function in *Oracle Spatial and Graph Developer's Guide*.

Example

The following example finds the centroid of all the U.S. Congressional district polygons.

```
SELECT centroid
FROM table(sem_match(
'select (orageo:aggrCentroid(?cgeom) as ?centroid)
 {?cdist orageo:hasExactGeometry ?cgeom } '
,sem_models('gov_all_vm'), null
,sem_aliases(
   sem_alias('usgovt','http://www.rdfabout.com/rdf/schema/usgovt/'),
   sem_alias('pol','http://www.rdfabout.com/rdf/schema/politico/'))
,null, null, ' ALLOW_DUP=T '));
```

B.2.2 orageo:aggrConvexHull

Format

orageo:aggrConvexhull(geom : geomLiteral) : ogc:wktLiteral

Description

Returns a geometry literal that is the convex hull of the group of specified geometry objects.. (The **convex hull** is a simple convex polygon that, for this funciton, completely encloses the group of geometry objects, using as few straight-line sides as possible to create the smallest polygon that completely encloses the geometry objects.)

Parameters

geom
Geometry objects. Specified as a query variable.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the SDO_AGGR_CONVEXHULL function in *Oracle Spatial and Graph Developer's Guide*.

Example

The following example finds the convex hull of all the U.S. Congressional district polygons.

```
SELECT chull
FROM table(sem_match(
'select (orageo:aggrConvexhull(?cgeom) as ?chull)
 {
   ?cdist orageo:hasExactGeometry ?cgeom } '
,sem_models('gov_all_vm'), null
,sem_aliases(
   sem_alias('usgovt','http://www.rdfabout.com/rdf/schema/usgovt/'),
```

```
   sem_alias('pol','http://www.rdfabout.com/rdf/schema/politico/'))
,null, null, ' ALLOW_DUP=T '));
```

B.2.3 orageo:aggrMBR

Format

orageo:aggrMBR(geom : geomLiteral) : ogc:wktLiteral

Description

Returns a geometry literal that is the minimum bounding rectangle (MBR) of the group of specified geometry objects.

Parameters

geom
Geometry objects. Specified as a query variable.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the SDO_AGGR_MBR function in *Oracle Spatial and Graph Developer's Guide*.

Example

The following example finds the MBR of all the U.S. Congressional district polygons.

```
SELECT mbr
FROM table(sem_match(
'select (orageo:aggrMBR(?cgeom) as ?mbr)
 {
   ?cdist orageo:hasExactGeometry ?cgeom } '
,sem_models('gov_all_vm'), null
,sem_aliases(
   sem_alias('usgovt','http://www.rdfabout.com/rdf/schema/usgovt/'),
   sem_alias('pol','http://www.rdfabout.com/rdf/schema/politico/'))
,null, null, ' ALLOW_DUP=T '));
```

B.2.4 orageo:aggrUnion

Format

orageo:aggrUnion(geom : geomLiteral) : ogc:wktLiteral

Description

Returns a geometry literal that is the topological union of the group of specified geometry objects.

Parameters

geom
Geometry objects. Specified as a query variable.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the SDO_GEOM.SDO_UNION function in *Oracle Spatial and Graph Developer's Guide*.

Example

The following example finds the union of all the U.S. Congressional district polygons.

```
SELECT u
FROM table(sem_match(
'select (orageo:aggrUnion(?cgeom) as ?u)
 {
   ?cdist orageo:hasExactGeometry ?cgeom } '
,sem_models('gov_all_vm'), null
,sem_aliases(
   sem_alias('usgovt','http://www.rdfabout.com/rdf/schema/usgovt/'),
   sem_alias('pol','http://www.rdfabout.com/rdf/schema/politico/'))
,null, null, ' ALLOW_DUP=T '));
```

B.2.5 orageo:area

Format

orageo:area(geom1 : geomLiteral, unit : Literal) : xsd:decimal

Description

Returns the area of `geom1` in terms of the specified unit of measure.

Parameters

geom1
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

unit
Unit of measurement: a quoted string with an SDO_UNIT value from the MDSYS.SDO_DIST_UNITS table (for example, `"unit=SQ_KM"`). See the section about unit of measurement support in *Oracle Spatial and Graph Developer's Guide* for more information about unit of measurement specification.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the SDO_GEOM.SDO_AREA function in *Oracle Spatial and Graph Developer's Guide*.

Example

The following example finds the U.S. Congressional district polygons with areas greater than 10,000 square kilometers.

```
SELECT name, cdist
FROM table(sem_match(
```

```
'{ ?person usgovt:name ?name .
   ?person pol:hasRole ?role .
   ?role pol:forOffice ?office .
   ?office pol:represents ?cdist .
   ?cdist orageo:hasExactGeometry ?cgeom
   FILTER (orageo:area(?cgeom, "unit=SQ_KM") > 10000) }'
,sem_models('gov_all_vm'), null
,sem_aliases(
   sem_alias('usgovt','http://www.rdfabout.com/rdf/schema/usgovt/'),
   sem_alias('pol','http://www.rdfabout.com/rdf/schema/politico/'))
,null, null, ' ALLOW_DUP=T '));
```

B.2.6 orageo:buffer

Format

orageo:buffer(geom1 : geomLiteral, distance : xsd:decimal, unit : Literal) : geomLiteral

Description

Returns a buffer polygon at a specified distance around or inside a geometry.

Parameters

geom1
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

distance
Distance value. Distance value. If the value is positive, the buffer is generated around `geom1`; if the value is negative (valid only for polygons), the buffer is generated inside `geom1`.

unit
Unit of measurement: a quoted string with an SDO_UNIT value from the MDSYS.SDO_DIST_UNITS table (for example, "`unit=KM`"). See the section about unit of measurement support in *Oracle Spatial and Graph Developer's Guide* for more information about unit of measurement specification.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the SDO_GEOM.SDO_BUFFER function in *Oracle Spatial and Graph Developer's Guide*.

Example

The following example finds the U.S. Congressional district polygons that are completely inside a 100-kilometer buffer around a specified point.

```
SELECT name, cdist
FROM table(sem_match(
'{ # HINT0={LEADING(?cgeom)}
   ?person usgovt:name ?name .
   ?person pol:hasRole ?role .
   ?role pol:forOffice ?office .
   ?office pol:represents ?cdist .
   ?cdist orageo:hasExactGeometry ?cgeom
   FILTER (
     orageo:relate(?cgeom,
```

```
        orageo:buffer("POINT(-71.46444 42.7575)"^^orageo:WKTLiteral,
                      100, "unit=KM"),
      "mask=inside")) }'
,sem_models('gov_all_vm'), null
,sem_aliases(
   sem_alias('usgovt','http://www.rdfabout.com/rdf/schema/usgovt/'),
   sem_alias('pol','http://www.rdfabout.com/rdf/schema/politico/'))
,null, null, ' ALLOW_DUP=T '));
```

B.2.7 orageo:centroid

Format

orageo:centroid(geom1 : geomLiteral) : geomLiteral

Description

Returns a point geometry that is the centroid of `geom1`. (The centroid is also known as the "center of gravity.")

Parameters

geom1
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

For an input geometry consisting of multiple objects, the result is weighted by the area of each polygon in the geometry objects. If the geometry objects are a mixture of polygons and points, the points are not used in the calculation of the centroid. If the geometry objects are all points, the points have equal weight.

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the SDO_GEOM.SDO_CENTROID function in *Oracle Spatial and Graph Developer's Guide*.

Example

The following example finds the U.S. Congressional district polygons with centroids within 200 kilometers of a specified point.

```
SELECT name, cdist
FROM table(sem_match(
'{ ?person usgovt:name ?name .
   ?person pol:hasRole ?role .
   ?role pol:forOffice ?office .
   ?office pol:represents ?cdist .
   ?cdist orageo:hasExactGeometry ?cgeom
   FILTER (orageo:withinDistance(orageo:centroid(?cgeom),
      "POINT(-71.46444 42.7575)"^^orageo:WKTLiteral,
      "distance=200 unit=KM")) } '
,sem_models('gov_all_vm'), null
,sem_aliases(
   sem_alias('usgovt','http://www.rdfabout.com/rdf/schema/usgovt/'),
   sem_alias('pol','http://www.rdfabout.com/rdf/schema/politico/'))
,null, null, ' ALLOW_DUP=T '));
```

B.2.8 orageo:convexHull

Format

orageo:convexHull(geom1 : geomLiteral) : geomLiteral

Description

Returns a polygon-type object that represents the convex hull of `geom1`. (The **convex hull** is a simple convex polygon that completely encloses the geometry object, using as few straight-line sides as possible to create the smallest polygon that completely encloses the geometry object.)

Parameters

geom1
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

A convex hull is a convenient way to get an approximation of a complex geometry object.

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the SDO_GEOM.SDO_CONVEX_HULL function in *Oracle Spatial and Graph Developer's Guide*.

Example

The following example finds the U.S. Congressional district polygons whose convex hull contains a specified point.

```
SELECT name, cdist
FROM table(sem_match(
'{ ?person usgovt:name ?name .
   ?person pol:hasRole ?role .
   ?role pol:forOffice ?office .
   ?office pol:represents ?cdist .
   ?cdist orageo:hasExactGeometry ?cgeom
   FILTER (orageo:relate(orageo:convexHull(?cgeom),
      "POINT(-71.46444 42.7575)"^^orageo:WKTLiteral,
      "mask=contains")) } '
,sem_models('gov_all_vm'), null
,sem_aliases(
   sem_alias('usgovt','http://www.rdfabout.com/rdf/schema/usgovt/'),
   sem_alias('pol','http://www.rdfabout.com/rdf/schema/politico/'))
,null, null, ' ALLOW_DUP=T '));
```

B.2.9 orageo:difference

Format

orageo:difference(geom1 : geomLiteral, geom2 : geomLiteral) : geomLiteral

Description

Returns a geometry object that is the topological difference (MINUS operation) of `geom1` and `geom2`.

Parameters

geom1
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

geom2
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the SDO_GEOM.SDO_DIFFERENCE function in *Oracle Spatial and Graph Developer's Guide*.

Example

The following example finds the U.S. Congressional district polygons whose centroid is inside the difference of two specified polygons.

```
SELECT name, cdist
FROM table(sem_match(
'{ ?person usgovt:name ?name .
   ?person pol:hasRole ?role .
   ?role pol:forOffice ?office .
   ?office pol:represents ?cdist .
   ?cdist orageo:hasExactGeometry ?cgeom
   FILTER (orageo:relate(orageo:centroid(?cgeom),
      orageo:difference(
                        "Polygon((-83.6 34.1, -83.2 34.1, -83.2 34.5, -83.6 34.5, -83.6
34.1))"^^orageo:WKTLiteral,
                        "Polygon((-83.2 34.3, -83.0 34.3, -83.0 34.5, -83.2 34.5, -83.2
34.3))"^^orageo:WKTLiteral),
      "mask=inside")) } '
,sem_models('gov_all_vm'), null
,sem_aliases(
   sem_alias('usgovt','http://www.rdfabout.com/rdf/schema/usgovt/'),
   sem_alias('pol','http://www.rdfabout.com/rdf/schema/politico/'))
,null, null, ' ALLOW_DUP=T '));
```

B.2.10 orageo:distance

Format

orageo:distance(geom1 : geomLiteral, geom2 : geomLiteral, unit : Literal) : xsd:decimal

Description

Returns the distance between the nearest pair of points or segments of `geom1` and `geom2` in terms of the specified unit of measure.

Parameters

geom1
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

geom2
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

unit
Unit of measurement: a quoted string with an SDO_UNIT value from the MDSYS.SDO_DIST_UNITS table (for example, `"unit=KM"`). See the section about unit of measurement support in *Oracle Spatial and Graph Developer's Guide* for more information about unit of measurement specification.

Usage Notes

Use orageo:withinDistance instead of `orageo:distance` whenever possible, because orageo:withinDistance has a more efficient index-based implementation.

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the SDO_GEOM.SDO_DISTANCE function in *Oracle Spatial and Graph Developer's Guide*.

Example

The following example finds the ten nearest U.S. Congressional districts to a specified point and orders them by distance from the point.

```
SELECT name, cdist
FROM table(sem_match(
'SELECT ?name ?cdist
 WHERE
 { # HINT0={LEADING(?cgeom)}
   ?person usgovt:name ?name .
   ?person pol:hasRole ?role .
   ?role pol:forOffice ?office .
   ?office pol:represents ?cdist .
   ?cdist orageo:hasExactGeometry ?cgeom
   FILTER (orageo:nearestNeighbor(?cgeom,
      "POINT(-71.46444 42.7575)"^^orageo:WKTLiteral,
      "sdo_num_res=10")) }
 ORDER BY ASC(orageo:distance(?cgeom,
                "POINT(-71.46444 42.7575)"^^orageo:WKTLiteral,
                "unit=KM"))'
,sem_models('gov_all_vm'), null
,sem_aliases(
   sem_alias('usgovt','http://www.rdfabout.com/rdf/schema/usgovt/'),
   sem_alias('pol','http://www.rdfabout.com/rdf/schema/politico/'))
,null, null, ' ALLOW_DUP=T '))
ORDER BY sem$rownum;
```

B.2.11 orageo:getSRID

Format

orageo:getSRID(geom : geomLiteral) : xsd:anyURI

Description

Returns the oracle spatial reference system (SRID) URI for `geom`.

Parameters

geom
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

Example

The following example finds spatial reference system URIs for U.S. Congressional district polygons.

```
SELECT csrid
FROM table(sem_match(
'SELECT (orageo:getSRID(?cgeom) AS ?csrid)
 WHERE
 { ?person usgovt:name ?name .
   ?person pol:hasRole ?role .
   ?role pol:forOffice ?office .
   ?office pol:represents ?cdist .
   ?cdist orageo:hasExactGeometry ?cgeom }'
,sem_models('gov_all_vm'), null
,sem_aliases(
   sem_alias('usgovt','http://www.rdfabout.com/rdf/schema/usgovt/'),
   sem_alias('pol','http://www.rdfabout.com/rdf/schema/politico/')
   )
,null, null, ' ALLOW_DUP=T '));
```

B.2.12 orageo:intersection

Format

orageo:intersection(geom1 : geomLiteral, geom2 : geomLiteral) : geomLiteral

Description

Returns a geometry object that is the topological intersection (AND operation) of `geom1` and `geom2`.

Parameters

geom1
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

geom2
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the SDO_GEOM.SDO_INTERSECTION function in *Oracle Spatial and Graph Developer's Guide*.

Example

The following example finds the U.S. Congressional district polygons whose centroid is inside the intersection of two specified polygons.

```
SELECT name, cdist
FROM table(sem_match(
'{ ?person usgovt:name ?name .
   ?person pol:hasRole ?role .
   ?role pol:forOffice ?office .
   ?office pol:represents ?cdist .
   ?cdist orageo:hasExactGeometry ?cgeom
   FILTER (orageo:relate(orageo:centroid(?cgeom),
      orageo:intersection(
                          "Polygon((-83.6 34.1, -83.2 34.1, -83.2 34.5, -83.6 34.5,
-83.6 34.1))"^^orageo:WKTLiteral,
                          "Polygon((-83.2 34.3, -83.0 34.3, -83.0 34.5, -83.2 34.5,
-83.2 34.3))"^^orageo:WKTLiteral),
      "mask=inside")) } '
,sem_models('gov_all_vm'), null
,sem_aliases(
   sem_alias('usgovt','http://www.rdfabout.com/rdf/schema/usgovt/'),
   sem_alias('pol','http://www.rdfabout.com/rdf/schema/politico/'))
,null, null, ' ALLOW_DUP=T '));
```

B.2.13 orageo:length

Format

orageo:length(geom1 : geomLiteral, unit : Literal) : xsd:decimal

Description

Returns the length or perimeter of `geom1` in terms of the specified unit of measure.

Parameters

geom1
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

unit
Unit of measurement: a quoted string with an SDO_UNIT value from the MDSYS.SDO_DIST_UNITS table (for example, `"unit=KM"`). See the section about unit of measurement support in *Oracle Spatial and Graph Developer's Guide* for more information about unit of measurement specification.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the SDO_GEOM.SDO_LENGTH function in *Oracle Spatial and Graph Developer's Guide*.

Example

The following example finds the U.S. Congressional district polygons with lengths (perimeters) greater than 1000 kilometers.

```
SELECT name, cdist
FROM table(sem_match(
'{ ?person usgovt:name ?name .
   ?person pol:hasRole ?role .
   ?role pol:forOffice ?office .
   ?office pol:represents ?cdist .
   ?cdist orageo:hasExactGeometry ?cgeom
   FILTER (orageo:legnth(?cgeom, "unit=KM") > 1000) }'
,sem_models('gov_all_vm'), null
,sem_aliases(
   sem_alias('usgovt','http://www.rdfabout.com/rdf/schema/usgovt/'),
   sem_alias('pol','http://www.rdfabout.com/rdf/schema/politico/'))
,null, null, ' ALLOW_DUP=T '));
```

B.2.14 orageo:mbr

Format

orageo:mbr(geom1 : geomLiteral) : geomLiteral

Description

Returns the minimum bounding rectangle of `geom1`, that is, the single rectangle that minimally encloses `geom1`.

Parameters

geom1
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the SDO_GEOM.SDO_MBR function in *Oracle Spatial and Graph Developer's Guide*.

Example

The following example finds the U.S. Congressional district polygons whose minimum bounding rectangle contains a specified point.

```
SELECT name, cdist
FROM table(sem_match(
'{ ?person usgovt:name ?name .
   ?person pol:hasRole ?role .
   ?role pol:forOffice ?office .
   ?office pol:represents ?cdist .
   ?cdist orageo:hasExactGeometry ?cgeom
   FILTER (orageo:relate(orageo:mbr(?cgeom),
      "POINT(-71.46444 42.7575)"^^orageo:WKTLiteral,
      "mask=contains")) } '
,sem_models('gov_all_vm'), null
,sem_aliases(
   sem_alias('usgovt','http://www.rdfabout.com/rdf/schema/usgovt/'),
```

```
   sem_alias('pol','http://www.rdfabout.com/rdf/schema/politico/'))
,null, null, ' ALLOW_DUP=T '));
```

B.2.15 orageo:nearestNeighbor

Format

orageo:nearestNeighbor(geom1: geomLiteral, geom2 : geomLiteral, param : Literal) : xsd:boolean

Description

Returns `true` if `geom1` is a nearest neighbor of `geom2`, where the size of the nearest neighbors set is specified by `param`; returns `false` otherwise.

Parameters

geom1
Geometry object. Specified as a query variable.

geom2
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

param
Determines the behavior of the operator. See the Usage Notes for the available keyword-value pairs.

Usage Notes

In the param parameter, the available keyword-value pairs are:

- `distance=`*`n`* specifies the maximum allowable distance for the nearest neighbor search.
- `sdo_num_res=`*`n`* specifies the size of the set for the nearest neighbor search.
- `unit=`*`unit`* specifies the unit of measurement to use with `distance` value. If you do not specify a value, the unit of measurement associated with the data is used.

`geom1` must be a local variable (that is, a variable that appears in the basic graph pattern that contains the `orageo:nearestNeighbor` spatial filter).

It is a good idea to use a `'LEADING(?var)' HINT0` hint when your query involves a restrictive `orageo:relate` spatial filter on `?var`.

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the SDO_NN operator in *Oracle Spatial and Graph Developer's Guide*.

Example

The following example finds the ten nearest U.S. Congressional districts to a specified point.

```
SELECT name, cdist
FROM table(sem_match(
'{ # HINT0={LEADING(?cgeom)}
   ?person usgovt:name ?name .
   ?person pol:hasRole ?role .
   ?role pol:forOffice ?office .
   ?office pol:represents ?cdist .
   ?cdist orageo:hasExactGeometry ?cgeom
```

```
   FILTER (orageo:nearestNeighbor(?cgeom,
      "POINT(-71.46444 42.7575)"^^orageo:WKTLiteral,
      "sdo_num_res=10")) } '
,sem_models('gov_all_vm'), null
,sem_aliases(
   sem_alias('usgovt','http://www.rdfabout.com/rdf/schema/usgovt/'),
   sem_alias('pol','http://www.rdfabout.com/rdf/schema/politico/'))
,null, null, ' ALLOW_DUP=T '));
```

B.2.16 orageo:relate

Format

orageo:relate(geom1: geomLiteral, geom2 : geomLiteral, param : Literal) : xsd:boolean

Description

Returns `true` if `geom1` and `geom2` satisfy the topological spatial relation specified by the `param` parameter; returns `false` otherwise.

Parameters

geom1
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

geom2
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

param
Specifies a list of mask relationships to check. See the list of keywords in the Usage Notes.

Usage Notes

The following `param` values (mask relationships) can be tested:

- ANYINTERACT: Returns TRUE if the objects are not disjoint.
- CONTAINS: Returns TRUE if the second object is entirely within the first object and the object boundaries do not touch; otherwise, returns FALSE.
- COVEREDBY: Returns TRUE if the first object is entirely within the second object and the object boundaries touch at one or more points; otherwise, returns FALSE.
- COVERS: Returns TRUE if the second object is entirely within the first object and the boundaries touch in one or more places; otherwise, returns FALSE.
- DISJOINT: Returns TRUE if the objects have no common boundary or interior points; otherwise, returns FALSE.
- EQUAL: Returns TRUE if the objects share every point of their boundaries and interior, including any holes in the objects; otherwise, returns FALSE.
- INSIDE: Returns TRUE if the first object is entirely within the second object and the object boundaries do not touch; otherwise, returns FALSE.
- ON: Returns ON if the boundary and interior of a line (the first object) is completely on the boundary of a polygon (the second object); otherwise, returns FALSE.
- OVERLAPBDYDISJOINT: Returns TRUE if the objects overlap, but their boundaries do not interact; otherwise, returns FALSE.

- OVERLAPBDYINTERSECT: Returns TRUE if the objects overlap, and their boundaries intersect in one or more places; otherwise, returns FALSE.
- TOUCH: Returns TRUE if the two objects share a common boundary point, but no interior points; otherwise, returns FALSE.

Values for `param` can be combined using the logical Boolean operator OR. For example, 'INSIDE + TOUCH' returns `TRUE` if the relationship between the geometries is INSIDE or TOUCH or both INSIDE and TOUCH; it returns `FALSE` if the relationship between the geometries is neither INSIDE nor TOUCH.

When invoking `orageo:relate` with a query variable and a constant geometry, always use the query variable as the first parameter and the constant geometry as the second parameter.

For best performance, `geom1` should be a local variable (that is, a variable that appears in the basic graph pattern that contains the `orageo:relate` spatial filter).

It is a good idea to use a `'LEADING(?var)' HINT0` hint when your query involves a restrictive `orageo:relate` spatial filter on `?var`.

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the SDO_RELATE operator in *Oracle Spatial and Graph Developer's Guide*.

Example

The following example finds the U.S. Congressional district that contains a specified point.

```
SELECT name, cdist
FROM table(sem_match(
'{ # HINT0={LEADING(?cgeom)}
   ?person usgovt:name ?name .
   ?person pol:hasRole ?role .
   ?role pol:forOffice ?office .
   ?office pol:represents ?cdist .
   ?cdist orageo:hasExactGeometry ?cgeom
   FILTER (orageo:relate(?cgeom,
      "POINT(-71.46444 42.7575)"^^orageo:WKTLiteral,
      "mask=contains")) } '
,sem_models('gov_all_vm'), null
,sem_aliases(
   sem_alias('usgovt','http://www.rdfabout.com/rdf/schema/usgovt/'),
   sem_alias('pol','http://www.rdfabout.com/rdf/schema/politico/'))
,null, null, ' ALLOW_DUP=T '
));
```

B.2.17 orageo:sdoDistJoin

Format

orageo:sdoDistJoin(geom1 : geomLiteral, geom2 : geomLiteral, param : Literal) : xsd:boolean

Description

Performs a spatial join based on distance between two geometries. Returns `true` if the distance between `geom1` and `geom2` is within the given value specified in `param`; returns `false` otherwise.

Parameters

geom1
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

geom2
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

param
Specifies a distance value and unit of measure to use for the distance-based spatial join. The distance value is added to the tolerance value of the associated spatial index. For example if `"distance=100 and unit=m"` is used with a tolerance value of 10 meters, then orageo:sdoDistJoin returns `true` if the distance between two geometries is no more than 110 meters.

Usage Notes

orageo:sdoDistJoin should be used when performing a distance-based spatial join between two large geometry collections. When performing a distance-based spatial join between one small geometry collection and one large geometry collection, invoking orageo:withinDistance with the small geometry collection as the first argument will usually give better performance than orageo:sdoDistJoin.

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the SDO_JOIN operator in *Oracle Spatial and Graph Developer's Guide*.

Example

The following example finds pairs of U.S. Congressional district polygons that are within 100 meters of each other.

```
SELECT cdist1, cdist2
FROM table(sem_match(
'{ ?cdist1 orageo:hasExactGeometry ?cgeom1 .
   ?cdist2 orageo:hasExactGeometry ?cgeom2
   FILTER (orageo:sdoDistJoin(?cgeom1, ?cgeom2,
      "distance=100 unit=m")) } '
,sem_models('gov_all_vm'), null
,sem_aliases(
   sem_alias('usgovt','http://www.rdfabout.com/rdf/schema/usgovt/'),
   sem_alias('pol','http://www.rdfabout.com/rdf/schema/politico/'))
,null, null, ' ALLOW_DUP=T '
));
```

B.2.18 orageo:sdoJoin

Format

orageo:sdoJoin(geom1 : geomLiteral, geom2 : geomLiteral, param : Literal) : xsd:boolean

Description

Performs a spatial join based on one or more topological relationships. Returns `true` if `geom1` and `geom2` satisfy the spatial relationship specified by `param`; returns `false` otherwise.

Parameters

geom1
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

geom2
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

param
Specifies a list of mask relationships to check. The topological relationship of interest.Valid values are 'mask=<*value*>' where <*value*> is one or more of the mask values that are valid for the SDO_RELATE operator (TOUCH, OVERLAPBDYDISJOINT, OVERLAPBDYINTERSECT, EQUAL, INSIDE, COVEREDBY, CONTAINS, COVERS, ANYINTERACT, ON). Multiple masks are combined with the logical Boolean operator OR (for example, "`mask=inside+touch`").

Usage Notes

orageo:sdoJoin should be used when performing a spatial join between two large geometry collections. When performing a spatial join between one small geometry collection and one large geometry collection, invoking orageo:relate with the small geometry collection as the first argument will usually give better performance than orageo:sdoJoin.

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the SDO_JOIN operator in *Oracle Spatial and Graph Developer's Guide*.

Example

The following example finds pairs of U.S. Congressional district polygons that have any spatial interaction.

```
SELECT cdist1, cdist2
FROM table(sem_match(
'{ ?cdist1 orageo:hasExactGeometry ?cgeom1 .
   ?cdist2 orageo:hasExactGeometry ?cgeom2
   FILTER (orageo:sdoJoin(?cgeom1, ?cgeom2,
      "mask=anyinteract")) } '
,sem_models('gov_all_vm'), null
,sem_aliases(
   sem_alias('usgovt','http://www.rdfabout.com/rdf/schema/usgovt/'),
   sem_alias('pol','http://www.rdfabout.com/rdf/schema/politico/'))
,null, null, ' ALLOW_DUP=T '
));
```

B.2.19 orageo:union

Format

orageo:union(geom1 : geomLiteral, geom2 : geomLiteral) : geomLiteral

Description

Returns a geometry object that is the topological union (OR operation) of `geom1` and `geom2`.

Parameters

geom1
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

geom2
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the SDO_GEOM.SDO_UNION function in *Oracle Spatial and Graph Developer's Guide*.

Example

The following example finds the U.S. Congressional district polygons whose centroid is inside the union of two specified polygons.

```
SELECT name, cdist
FROM table(sem_match(
'{ ?person usgovt:name ?name .
   ?person pol:hasRole ?role .
   ?role pol:forOffice ?office .
   ?office pol:represents ?cdist .
   ?cdist orageo:hasExactGeometry ?cgeom
   FILTER (orageo:relate(orageo:centroid(?cgeom),
      orageo:union("Polygon((-83.6 34.1, -83.2 34.1, -83.2 34.5, -83.6 34.5, -83.6
34.1))"^^orageo:WKTLiteral,
                   "Polygon((-83.2 34.3, -83.0 34.3, -83.0 34.5, -83.2 34.5, -83.2
34.3))"^^orageo:WKTLiteral),
      "mask=inside")) } '
,sem_models('gov_all_vm'), null
,sem_aliases(
   sem_alias('usgovt','http://www.rdfabout.com/rdf/schema/usgovt/'),
   sem_alias('pol','http://www.rdfabout.com/rdf/schema/politico/'))
,null, null, ' ALLOW_DUP=T '));
```

B.2.20 orageo:withinDistance

Format

orageo:withinDistance(geom1 : geomLiteral, geom2 : geomLiteral, distance : xsd:decimal, unit : Literal) : xsd:boolean

Description

Returns `true` if the distance between `geom1` and `geom2` is less than or equal to `distance` when measured in `unit`; returns `false` otherwise.

Parameters

geom1
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

geom2
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

distance
Distance value.

unit
Unit of measurement: a quoted string with an SDO_UNIT value from the MDSYS.SDO_DIST_UNITS table (for example, "`unit=KM`"). See the section about unit of measurement support in *Oracle Spatial and Graph Developer's Guide* for more information about unit of measurement specification.

Usage Notes

When invoking this function with a query variable and a constant geometry, always use the query variable as the first parameter and the constant geometry as the second parameter.

For best performance, `geom1` should be a local variable (that is, a variable that appears in the basic graph pattern that contains the `orageo:withinDistance` spatial filter).

It is a good idea to use a `'LEADING(?var)' HINT0` hint when your query involves a restrictive `orageo:withinDistance` spatial filter on `?var`.

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the SDO_WITHIN_DISTANCE operator in *Oracle Spatial and Graph Developer's Guide*.

Example

The following example finds the U.S. Congressional districts that are within 100 kilometers of a specified point.

```
SELECT name, cdist
FROM table(sem_match(
'{ # HINT0={LEADING(?cgeom)}
   ?person usgovt:name ?name .
   ?person pol:hasRole ?role .
   ?role pol:forOffice ?office .
   ?office pol:represents ?cdist .
   ?cdist orageo:hasExactGeometry ?cgeom
   FILTER (orageo:withinDistance(?cgeom,
      "POINT(-71.46444 42.7575)"^^orageo:WKTLiteral,
      100, "KM")) } '
,sem_models('gov_all_vm'), null
,sem_aliases(
   sem_alias('usgovt','http://www.rdfabout.com/rdf/schema/usgovt/'),
   sem_alias('pol','http://www.rdfabout.com/rdf/schema/politico/'))
,null, null, ' ALLOW_DUP=T '));
```

B.2.21 orageo:xor

Format

orageo:xor(geom1 : geomLiteral, geom2 : geomLiteral) : geomLiteral

Description

Returns a geometry object that is the topological symmetric difference (XOR operation) of `geom1` and `geom2`.

Parameters

geom1
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

geom2
Geometry object. Specified as a query variable or a constant `geomLiteral` value.

Usage Notes

See Spatial Support for information about representing, indexing, and querying spatial data in RDF.

See also the SDO_GEOM.SDO_XOR function in *Oracle Spatial and Graph Developer's Guide*.

Example

The following example finds the U.S. Congressional district polygons whose centroid is inside the symmetric difference of two specified polygons.

```
SELECT name, cdist
FROM table(sem_match(
'{ ?person usgovt:name ?name .
   ?person pol:hasRole ?role .
   ?role pol:forOffice ?office .
   ?office pol:represents ?cdist .
   ?cdist orageo:hasExactGeometry ?cgeom
   FILTER (orageo:relate(orageo:centroid(?cgeom),
      orageo:xor(
           "Polygon((-83.6 34.1, -83.2 34.1, -83.2 34.5, -83.6 34.5, -83.6
34.1))"^^orageo:WKTLiteral,
           "Polygon((-83.2 34.3, -83.0 34.3, -83.0 34.5, -83.2 34.5, -83.2
34.3))"^^orageo:WKTLiteral),
      "mask=inside")) } '
,sem_models('gov_all_vm'), null
,sem_aliases(
   sem_alias('usgovt','http://www.rdfabout.com/rdf/schema/usgovt/'),
   sem_alias('pol','http://www.rdfabout.com/rdf/schema/politico/'))
,null, null, ' ALLOW_DUP=T '));
```

C

RDF Support in SQL Developer

You can use Oracle SQL Developer to perform operations related to the RDF Graph feature of Oracle Graph.

- About RDF Support in SQL Developer
  The RDF support in SQL Developer is available through the Connections navigator.
- Setting Up the RDF Semantic Graph Support In SQL Developer
  This section applies only if you are using Oracle Database 19c or later. You must execute a setup procedure to enable RDF Semantic Graph support in SQL Developer for schema-private networks only.
- Working with RDF Semantic Networks Using SQL Developer
  You can create an RDF semantic network to work with RDF data using SQL Developer.
- Bulk Loading RDF Data Using SQL Developer
  RDF Bulk load operations can be invoked from SQL Developer.

C.1 About RDF Support in SQL Developer

The RDF support in SQL Developer is available through the Connections navigator.

You can use SQL Developer to create and manage RDF-related objects in an Oracle database. Oracle Graph support for semantic technologies consists mainly of Resource Description Framework (RDF) and a subset of the Web Ontology Language (OWL). These capabilities are referred to as the RDF Knowledge Graph feature of Oracle Graph.

Support for SQL Developer is included in RDF if the following conditions are true:

- The database connection is to Oracle Database release 12.1 or later.
- RDF semantic graph support is enabled in the database. After this support is enabled, the SDO_RDF_TRIPLE_S type will be available.

If you expand an Oracle Database connection that meets these conditions, near the bottom of the child nodes for the connection is **RDF Semantic Graph**.

C.2 Setting Up the RDF Semantic Graph Support In SQL Developer

This section applies only if you are using Oracle Database 19c or later. You must execute a setup procedure to enable RDF Semantic Graph support in SQL Developer for schema-private networks only.

> **Note:**
>
> This setup is not required for semantic networks in MDSYS schema. Starting from Oracle Database 19c, it is always recommended to create semantic networks in database user schemas.

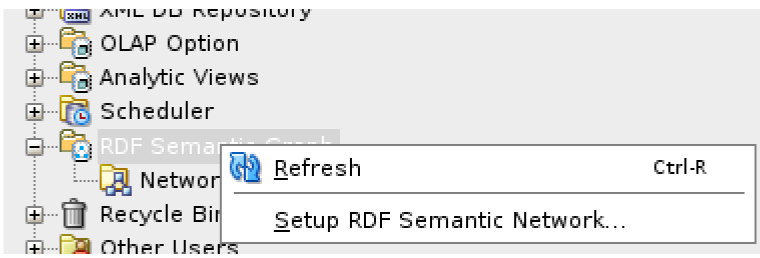
Running this setup creates helper functions that are needed to populate RDF network dictionary information in SQL Developer.

> **Note:**
>
> If you do not perform this one-time setup procedure, you may encounter an error when trying to expand RDF network metadata nodes (such as `REGULAR_MODELS`, `RDF_VIEWS`, `RULEBASES`, and so on) in SQL Developer.

To perform this setup:

1. Open SQL Developer.
2. Right-click the **RDF Semantic Graph** node and select **Setup RDF Semantic Graph** to execute the one-time setup procedure.

Figure C-1 RDF Semantic Graph Setup



The following table helps you to determine if you require a DBA privilege to have this option available.

Table C-1 RDF Semantic Graph Setup Specific To SQL Developer and Oracle DB Version

Oracle DB Version	SQL Developer Version	Type of User	Expected Result
19c or later	Earlier to 20.3	To be executed once by a user with DBA privilege	Required types and functions are installed in MDSYS schema.

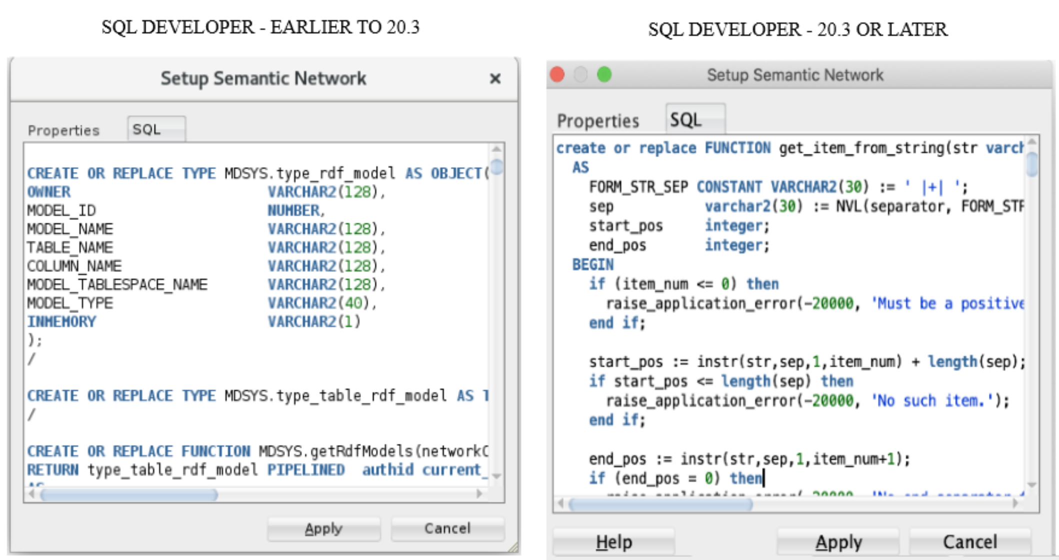
Table C-1 (Cont.) RDF Semantic Graph Setup Specific To SQL Developer and Oracle DB Version

Oracle DB Version	SQL Developer Version	Type of User	Expected Result
19c or later	20.3 or later	To be executed once individually by each user	Required types and functions are installed in the user's schema.

> **Note:**
>
> If you have already set up the RDF Semantic Graph support in Oracle Database Release 19c or later with a SQL Developer version *earlier* than 20.3, but you have started using SQL Developer Release 20.3 or later, then you will need to perform the setup again, because the metadata functions are different from previous ones that were installed in the MDSYS schema.

3. Click **Apply**.
   Optionally, you can also click the **SQL** tab to view the procedure.

Figure C-2 Apply RDF Semantic Graph Setup



The required types and function are installed in the appropriate schema. Once this setup is executed, the **RDF Semantic Graph** option appears grayed out.

C.3 Working with RDF Semantic Networks Using SQL Developer

You can create an RDF semantic network to work with RDF data using SQL Developer.

You can view the available networks in the database schema associated with your connection by expanding the Networks node in the RDF Semantic Graph tree.

From Release 19c onwards, an RDF semantic network is supported in both user schema and MDSYS schema. See the following table to determine the semantic network type recommended for you depending on your database version.

Table C-2 Recommended Semantic Network Type

Database Release	Supported Network(s)	Recommended Network
18c or earlier	All RDF metadata belongs only to MDSYS Network.	MDSYS Network
19c or later	• MDSYS Network • Schema-Private Network	Schema-Private Network

- Creating an RDF Semantic Network Using SQL Developer
  Under the Networks node, you can create one or more RDF semantic networks.
- Refreshing Semantic Network Indexes Using SQL Developer
  RDF uses semantic network indexes (some created automatically), which you can refresh.
- Gathering RDF Statistics Using SQL Developer
  You can gather statistics about RDF and OWL tables and their indexes.
- Purging Unused Values from a Network Using SQL Developer
  You can purge unused (invalid) geometry literal values from the semantic network.
- Dropping a Semantic Network Using SQL Developer
  Dropping a semantic network removes structures used for persistent storage of semantic data..

C.3.1 Creating an RDF Semantic Network Using SQL Developer

Under the Networks node, you can create one or more RDF semantic networks.

To create a new semantic network:

1. Right-click **Networks** and select **Create Semantic Network**.
   This operation is available for users depending on the Oracle Database version and the SQL Developer version used. See the following table for more information:

Table C-3 Release Specific Instructions to Create a Semantic Network

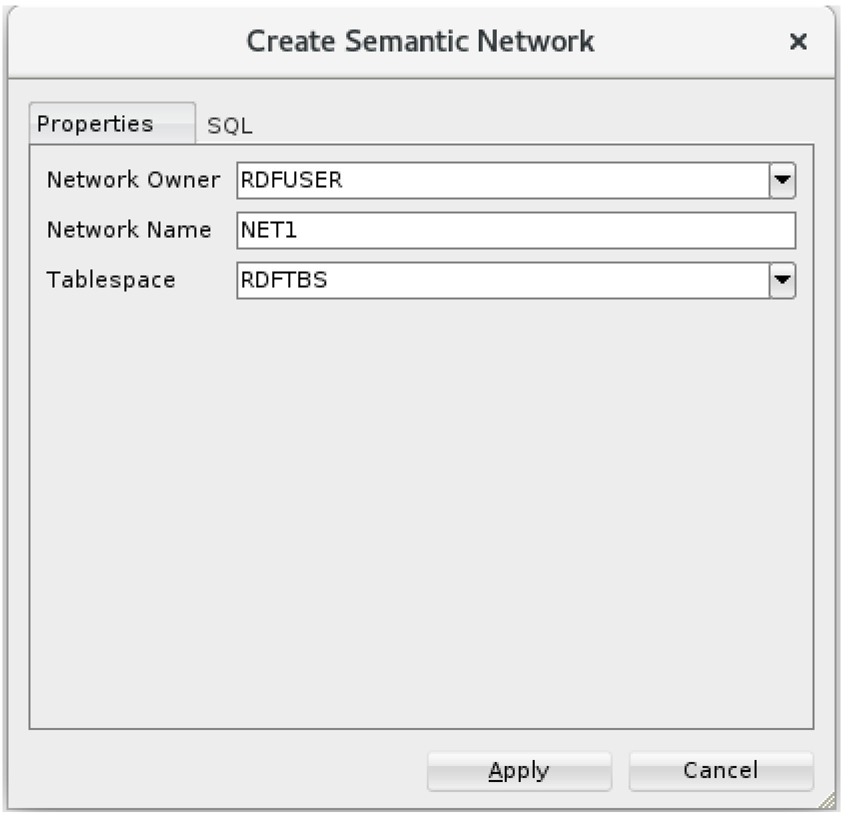
Oracle DB Release	SQL Developer Version	User Requirement
18c or earlier	Any	Only a user having a DBA role can create an MDSYS network.
For Release 19c- prior 19.3	Any	Only a user having a DBA role can create a schema-private network.

Table C-3 (Cont.) Release Specific Instructions to Create a Semantic Network

Oracle DB Release	SQL Developer Version	User Requirement
19.3 or later	Prior 20.3	Only a user having a DBA role can create a schema-private network.
19.3 or later	20.3 or later	Any database user can create a schema-private network directly.

Create Semantic Network window opens as shown:

Figure C-3 Create Semantic Network



2. Select a **Network Owner**, that is, the database schema that will be the owner of the network.
   - For release 18c and earlier, the owner is always MDSYS.
   - For release 19c before 19.3, select the network owner.
   - For release 19.3 and later, the network owner is always the connection user schema.
3. Enter a **Network Name**.

> **Note:**
>
> For release 18c and earlier, this field is blank and not editable.

4. Select a **Tablespace** to be associated with the network. (If the tablespace or tablespaces necessary for semantic networks do not already exist, see Creating Tablespaces for Semantic Networks Using SQL Developer.)

5. Click **Apply**.
   The RDF semantic network is created.

   You can verify the RDF semantic network creation by viewing the following child nodes under the created Nework:

   - REGULAR_MODELS
   - VIRTUAL_MODELS
   - RDF_VIEWS
   - RULEBASES
   - ENTAILMENTS
   - NETWORK_INDEXES (RDF_LINK$)
   - DATATYPE_INDEXES (RDF_VALUE$)
   - BULK_LOAD_TRACES

You can now perform the following operations on each created network:

- Gather Statistics
- Refresh semantic network indexes
- Purge unused values
- Drop semantic network

- Creating Tablespaces for Semantic Networks Using SQL Developer
  If the tablespace or tablespaces required for semantic networks do not already exist, you can create them.

C.3.1.1 Creating Tablespaces for Semantic Networks Using SQL Developer

If the tablespace or tablespaces required for semantic networks do not already exist, you can create them.

You can adjust those that were created automatically as part of the semantic network setup operation.

The recommended practice is to use three tablespaces for RDF Semantic Graph:

- Tablespace for RDF storage (create a new tablespace named RDFTBS)
- Tablespace for temporary data (create a new tablespace named TEMPTBS)
- Tablespace for other user data (use the existing tablespace named USERS)

In the DBA navigator (***not*** the Connections navigator), for the `system` connection click **Storage**, then **Tablespaces**. For the new tablespaces (right-click and select **Create New**), and select any desired name (the ones listed here are just examples). Accept default values or specified desired options.

1. Create RDFTBS for storing RDF data.

   Name (tablespace name): `RDFTBS`

   Tablespace Type: `Permanent`

   Under File Specification, **Name**: `'RDFTBS.DBF'`

   Directory: Desired file system directory. For example: `/u01/app/oracle/oradata/orcl12c/orcl`

File Size: Desired file initial size. For example: `1 G`

Check **Reuse** and **Auto Extend On**.

Next Size: Desired size of each extension increment. For example: `512 M`

Max Size: Desired file maximum size. For example: `10 G`

Click **OK**.

2. Create TEMPTBS for temporary work space.

   Right-click and select **Create New**.

   Name (tablespace name): `TEMPTBS`

   Tablespace Type: `Temporary`

   Under File Specification, **Name**: `'TEMPTBS.DBF'`

   Directory: Desired file system directory. For example: `/u01/app/oracle/oradata/orcl12c/orcl`

   File Size: Desired file initial size. For example: `1 G`

   Check **Reuse** and **Auto Extend On**.

   Next Size: Desired size of each extension increment. For example: `256 M`

   Max Size: Desired file maximum size. For example: `8 G`

3. Make TEMPTBS the default temporary tablespace for the database, by using the SQL Worksheet for the `system` connection's SQL Worksheet to execute the following statement:

```
SQL> alter database default temporary tablespace TEMPTBS;
```

C.3.2 Refreshing Semantic Network Indexes Using SQL Developer

RDF uses semantic network indexes (some created automatically), which you can refresh.

You can create additional semantic indexes if you wish, and you can adjust those that were created automatically.

There are multicolumn B-Tree semantic indexes over the following columns:

- S - subject
- P - predicate
- C - canonical object
- G - graph
- M - model

Two indexes are created by default: PCSGM and PSCGM. However, you can use a three-index setup to better cover more combinations of S, P, and C: PSCGM, SPCGM, and CSPGM.

In the Connections navigator (***not*** the DBA navigator), expand the `system` connection, expand **RDF Semantic Graph**, then click **Network Indexes (RDF_LINK)**.

1. Add the SPCGM index.

   Right-click and select **Create Semantic Index**. Suggested **Index code**: `SPCGM`

   Click **OK**.

2. Add the CSPGM index.

   Right-click and select **Create Semantic Index**. Suggested **Index code**: `CSPGM`

   Click **OK**.

3. Drop the PSCGM index.

   Right-click `RDF_LINK_PSCGM_IDX` and select **Drop Semantic Index**.

The result will be these three indexes:

- RDF_LINK_PSCGM_IDX
- RDF_LINK_SPCGM_IDX
- RDF_LINK_CSPGM_IDX

C.3.3 Gathering RDF Statistics Using SQL Developer

You can gather statistics about RDF and OWL tables and their indexes.

To gather statistics about a semantic network, right-click the network name and select **Gather Statistics**.

The following parameters can be defined in the dialog box:

Network Owner: The connection user (not editable).

Network Name: Name of the network (not editable).

Just on Values: If enabled (checked), collects statistics only on the table containing the lexical values of triples. If not enabled (unchecked), collects statistics on all major tables related to the storage of RDF and OWL data.

Degree of Parallelism: Number of parallel execution servers associated with the operation.

To complete the network creation, click **Apply**.

C.3.4 Purging Unused Values from a Network Using SQL Developer

You can purge unused (invalid) geometry literal values from the semantic network.

Deletion of triples over time may lead to a subset of the values in the RDF_VALUE$ table becoming unused in any of the RDF triples or rules currently in the semantic network. To delete such unused values from the RDF_VALUE$ table, right-click the network name and select **Purge Unused Values**..

The following parameters can be defined in the dialog box:

Network Owner: The connection user (not editable).

Network Name: Name of the network (not editable).

MBV_METHOD=SHADOW: If enabled (checked), may result faster processing when a large number of values need to be purged.

Degree of Parallelism: Number of parallel execution servers associated with the operation.

PUV_COMPUTE_VIDS_USED: If enabled (checked), may result faster processing when most of the values are expected to be purged.

Extra Flags: Specify any additional keywords and values to be added in the `flags` parameter for the SEM_APIS.PURGE_UNUSED_VALUES procedure that will be executed (click the SQL tab to see the complete SQL statement).

To perform the operation, click **Apply**.

C.3.5 Dropping a Semantic Network Using SQL Developer

Dropping a semantic network removes structures used for persistent storage of semantic data..

To drop a semantic network, right-click the network name and select **Drop Semantic Network**.

The following parameters can be defined in the dialog box:

Network Owner: The connection user (not editable).

Network Name: Name of the network (not editable).

Cascade: If enabled (checked), also drops any existing semantic technology models and rulebases for the network, and removes structures used for persistent storage of semantic data for the network. If not enabled (unchecked), the operation will fail if any semantic technology models or rulebases exist in the network.

To perform the operation, click **Apply**.

C.4 Bulk Loading RDF Data Using SQL Developer

RDF Bulk load operations can be invoked from SQL Developer.

Two major steps are required after some initial preparation: (1) loading data from the file system into a “staging“ table and (2) loading data from a “staging“ table into a semantic model.

Do the following to prepare for the actual bulk loading.

1. Prepare the RDF dataset or datasets.
   - The data must be on the file system of the Database server – not on the client system.
   - The data must be in N-triple or N-quad format. (Apache Jena, for example, can be used to convert other formats to N-triple/N-quad,)
   - A Unix named pipe can be used to decompress zipped files on the fly.

   For example, you can download RDF datasets from LinkedGeoData. For an introduction, see http://linkedgeodata.org/Datasets and http://linkedgeodata.org/RDFMapping.

   To download from LinkedGeoData, go to https://hobbitdata.informatik.uni-leipzig.de/LinkedGeoData/downloads.linkedgeodata.org/releases/ and browse the listed directories. For a fairly small dataset you can download https://hobbitdata.informatik.uni-leipzig.de/LinkedGeoData/downloads.linkedgeodata.org/releases/2014-09-09/2014-09-09-ontology.sorted.nt.bz2.

   Each .bz2 file is a compressed archive containing a comparable-named .nt file. To specify an .nt file as a data source, you must extract (decompress) the corresponding .bz2 file, unless you create a Unix named pipe to avoid having to store uncompressed data.

2. Create a regular, non-DBA user to perform the load.

   For example, using the DBA navigator (***not*** the Connections navigator), expand the `system` connection, expand **Security**, right-click **Users**, and select **Create New**.

   Create a user (for example, named `RDFUSER`) with CONNECT, RESOURCE, and UNLIMITED TABLESPACE privileges.

3. Add a connection for this regular, non-DBA user (for example, a connection named `RDFUSER`).

   Default Tablespace: `USERS`

   Temporary Tablespace: `TEMPTBS`

4. As the system user, create a directory in the database that points to your RDF data directory.

   Using the Connections navigator (***not*** the DBA navigator), expand the `system` connection, right-click **Directory** and select **Create Directory**.

   Directory Name: Desired directory name. For example: `RDFDIR`

   Database Server Directory: Desired location for the directory. For example: `/home/oracle/RDF/MyData`

   Click **Apply**.

5. Grant privileges on the directory to the regular, non-DBA user (for example, RDFUSER). For example, using the `system` connection's SQL Worksheet:

   ```
   SQL> grant read, write on directory RDFDIR to RDFUSER;
   ```

   Tip: you can use a named pipe to avoid having to store uncompressed data. For example:

   ```
   $ mkfifo named_pipe.nt
   $ bzcat myRdfFile.nt.bz2 > named_pipe.nt
   ```

6. Expand the regular, non-DBA user (for example, `RDFUSER`) connection and click **RDF Semantic Graph**.

7. Create a model to hold the RDF data.

   Click **Model**, then **New Model**.

   Model Name: Enter a model name (for example, `MY_ONTOLOGY`)

   Application Table: `* Create new <Model_Name>_TPL table *` (that is, have an application table with a triple column named TRIPLE automatically created)

   Model Tablespace: tablespace to hold the RDF data (for example, `RDFTBS`)

   Click **Apply**.

   To see the model, expand **Models** in the object hierarchy, and click the model name to bring up the SPARQL editor for that model.

   You can run a query and see that the model is empty.

Using the Models menu, perform a bulk load from the Models menu. Bulk load has two phases:

- Loading data from the file system into a simple "staging" table in the database. This uses an external table to read from the file system.
- Loading data from the staging table into the semantic network. Load from the staging table into the model (for example, `MY_ONTOLOGY`).

To perform these two phases:

1. Load data into the staging table.

   Right-click `REGULAR_MODELS` (under the network name) and select **Load RDF Data into Staging Table from External Table**.

For Source External Table, **Source Table**: Desired table name (for example, `MY_ONTOLOGY_EXT`).

Log File: Desired file name (for example, `my_ontology.log`)

Bad File: Desired file name (for example, `my_ontology.bad`)

Source Table Owner: Schema of the table with RDF data (for example, `RDFUSER`)

For Input Files, **Input Files**: Input file (for example, `named_pipe.nt`).

For Staging Table, **Staging table**: Name for the staging table (for example, `MY_ONTOLOGY_STAGE`).

If the table does not exist, check **Create Staging Table**.

Input Format: Desired format (for example, `N-QUAD`)

Staging Table Owner: Schema for the staging table (for example, `RDFUSER`)

2. Load from the staging table into the model.

> **Note:**
>
> Unicode data in the staging table should be escaped as specified in WC3 N-Triples format (http://www.w3.org/TR/rdf-testcases/#ntriples). You can use the SEM_APIS.ESCAPE_RDF_TERM function to escape Unicode values in the staging table. For example:
>
> ```
> create table esc_stage_tab(rdf$stc_sub, rdf$stc_pred, rdf$stc_obj);
>
> insert /*+ append nologging parallel */ into esc_stage_tab
> (rdf$stc_sub, rdf$stc_pred, rdf$stc_obj)
> select sem_apis.escape_rdf_term(rdf$stc_sub, options=>' UNI_ONLY=T
> '), sem_apis.escape_rdf_term(rdf$stc_pred, options=>' UNI_ONLY=T
> '), sem_apis.escape_rdf_term(rdf$stc_obj, options=>' UNI_ONLY=T ')
> from stage_tab;
> ```

Right-click `REGULAR_MODELS` (under the network name) and select **Bulk Load into Model from staging Table**.

Model: Name for the model (for example, `MY_ONTOLOGY`).

(If the model does not exist, check **Create Model**. However, in this example, the model does already exist.)

Staging Table Owner: Schema of the staging table (for example, `RDFUSER`)

Staging Table Name: Name of the staging table (for example, `MY_ONTOLOGY_STAGE`)

Parallel: Degree of parallelism (for example, `2`)

Suggestion: Check the following options: **MBV_METHOD=SHADOW**, **Rebuild application table indexes**, **Create event trace table**

Click **Apply**.

Do the following after the bulk load operation.

1. Gather statistics for the whole semantic network.

In the Connections navigator for a DBA user, expand the RDF Semantic Graph node for the connection and select **Gather Statistics (DBA)**).

2. Run some SPARQL queries on our model.

   In the Connections navigator, expand the RDF Semantic Graph node for the connection and select the model.

   Use the SPARQL Query Editor to enter and execute desired SPARQL queries.

3. Optionally, check the bulk load trace to get information about each step.

   Expand RDF Semantic Graph and then expand **Bulk Load Traces** to display a list of bulk load traces. Clicking one of them will show useful information about the execution time for the load, number of distinct values and triples, number of duplicate triples, and other details.

D
MDSYS-Owned Semantic Network

A semantic network can be created in and owned by the MDSYS schema.

If a network is created in the MDSYS schema, it is an unnamed semantic network available to the entire database.

D.1 Creating an MDSYS-owned Semantic Network

You can create an MDSYS-owned semantic network using a SQL based interface such as SQL Developer, SQLPLUS, or from a Java program using JDBC.

1. Connect to **Oracle Database** as a `SYSTEM` user with a DBA privilege.

   ```
   CONNECT system/<password-for-system-user>
   ```

2. Create a **tablespace** for storing the **RDF graphs**. Use a suitable operating system folder and filename.

   ```
   CREATE TABLESPACE rdftbs
     DATAFILE 'rdftbs.dat'
     SIZE 128M REUSE
     AUTOEXTEND ON NEXT 64M
     MAXSIZE UNLIMITED
     SEGMENT SPACE MANAGEMENT AUTO;
   ```

3. Grant quota on `rdftbs` to MDSYS.

   ```
   ALTER USER MDSYS QUOTA UNLIMITED ON rdftbs;
   ```

4. Create a **tablespace** for storing the **user data**. Use a suitable operating system folder and filename.

```
CREATE TABLESPACE usertbs
  DATAFILE 'usertbs.dat'
  SIZE 128M REUSE
  AUTOEXTEND ON NEXT 64M
  MAXSIZE UNLIMITED
  SEGMENT SPACE MANAGEMENT AUTO;
```

5. Create a database **user** to create or use RDF graphs or do both using the adapter.

```
CREATE USER rdfuser
IDENTIFIED BY <password-for-rdfuser>
DEFAULT TABLESPACE usertbs
QUOTA 5G ON usertbs;
```

6. Grant quota on `rdftbs` to RDFUSER.

```
ALTER USER RDFUSER QUOTA 5G ON rdftbs;
```

7. Grant the necessary **privileges** to the new database user.

```
GRANT CONNECT, RESOURCE TO rdfuser;
```

8. Create an MDSYS-owned **semantic network**.

```
EXECUTE SEM_APIS.CREATE_SEM_NETWORK(tablespace_name =>'rdftbs');
```

9. Verify that MDSYS-owned semantic network has been created successfully.

```
SELECT table_name
  FROM sys.all_tables
  WHERE table_name = 'RDF_VALUE$' AND owner='MDSYS';
```

Presence of `RDF_VALUE$` table in the MDSYS schema shows that the MDSYS-owned semantic network has been created successfully.

```
TABLE_NAME
-----------
RDF_VALUE$
```

D.2 Getting Started with Semantic Data in an MDSYS-Owned Network

Get started working with semantic data in an MDSYS-owned network.

1. Create a tablespace for the system tables. You must be connected as a user with appropriate privileges to create the tablespace. The following example creates a tablespace named `rdf_tblspace`:

```
CREATE TABLESPACE rdf_tblspace
 DATAFILE 'rdf_tblspace.dat' SIZE 1024M REUSE
 AUTOEXTEND ON NEXT 256M MAXSIZE UNLIMITED
 SEGMENT SPACE MANAGEMENT AUTO;
```

2. Create an MDSYS-owned semantic network.

   Creating a semantic network adds semantic data support to an Oracle database. You must create a semantic network as a user with DBA privileges.

   The following example creates a MDSYS-owned semantic network:

```
EXECUTE SEM_APIS.CREATE_SEM_NETWORK('rdf_tblspace');
```

3. Create a database user under whose schema you will manage your semantic data and grant the necessary privileges to the database user. You must be connected as a user with appropriate privileges to create the database user.

   The following example creates a database user `rdfuser` and grants the necessary privileges to `rdfuser`:

```
CREATE USER rdfuser
IDENTIFIED BY <password-for-rdfuser>
QUOTA 5G ON rdf_tblspace;

GRANT CONNECT, RESOURCE, CREATE VIEW TO rdfuser;
```

4. Connect as the database user.

```
CONNECT rdfuser/<password-for-rdfuser>
```

> **Note:**
>
> You must not perform the following steps while connected as SYS, SYSTEM, or MDSYS.

5. Create an application table to store references to the semantic data and manage privileges for insert, update and delete operations. (You do not need to be connected as a user with DBA privileges for this step and the remaining steps.)

   This table must contain a column of type `SDO_RDF_TRIPLE_S`, which will contain references to all data associated with a single model.

   The following example creates a table named `articles_rdf_data` with one column to hold the data for triples:

```
CREATE TABLE articles_rdf_data (triple SDO_RDF_TRIPLE_S) COMPRESS;
```

6. Create a model.

   When you create a model, you must specify the model name, the table to hold references to semantic data for the model, and the column of type SDO_RDF_TRIPLE_S in that table.

The following command creates a model named `articles` in the MDSYS-owned network, which will use the table created in the preceding step.

```
EXECUTE SEM_APIS.CREATE_SEM_MODEL('articles', 'articles_rdf_data',
'triple');
```

After you create the model, you can insert triples into the model, as shown in the examples in Semantic Data Examples (PL/SQL and Java).

> **Note:**
>
> You must omit the `network_owner` and `network_name` arguments in the Semantic Data Examples (PL/SQL and Java) when using an MDSYS-owned semantic network.

D.3 Example Queries Using Graph Support for Apache Jena

This section describes example queries using the support for Apache Jena and is based on the RDF metadata that is stored in the MDSYS schema.

To run a query, you must do the following:

1. Include the code in a Java source file. The examples used in this section are supplied in files in the `examples` directory of the support for Apache Jena download.
2. Compile the Java source file. For example:

   ```
   > javac -classpath ../jar/'*' Test.java
   ```

3. Run the compiled file. For example:

   ```
   > java -classpath ./:../jar/'*'  Test jdbc:oracle:thin:@localhost:1521:orcl scott
   <password-for-scott> M1
   ```

- Test.java: Query Family Relationships
- Test6.java: Load OWL Ontology and Perform OWLPrime inference
- Test7.java: Bulk Load OWL Ontology and Perform OWLPrime inference
- Test8.java: SPARQL OPTIONAL Query
- Test9.java: SPARQL Query with LIMIT and OFFSET
- Test10.java: SPARQL Query with TIMEOUT and DOP
- Test11.java: Query Involving Named Graphs
- Test12.java: SPARQL ASK Query
- Test13.java: SPARQL DESCRIBE Query
- Test14.java: SPARQL CONSTRUCT Query
- Test15.java: Query Multiple Models and Specify "Allow Duplicates"
- Test16.java: SPARQL Update
- Test17.java: SPARQL Query with ARQ Built-In Functions
- Test18.java: SELECT Cast Query
- Test19.java: Instantiate Oracle Database Using OracleConnection
- Test20.java: Oracle Database Connection Pooling

D.3.1 Test.java: Query Family Relationships

Example D-1 Query Family Relationships

Example D-1 specifies that John is the father of Mary, and it selects and displays the subject and object in each `fatherOf` relationship

```
import oracle.spatial.rdf.client.jena.*;
import org.apache.jena.rdf.model.Model;
import org.apache.jena.graph.*;
import org.apache.jena.query.*;
public class Test {

  public static void main(String[] args) throws Exception
  {
    String szJdbcURL = args[0];
    String szUser    = args[1];
    String szPasswd  = args[2];
    String szModelName = args[3];

    Oracle oracle = new Oracle(szJdbcURL, szUser, szPasswd);
    Model model = ModelOracleSem.createOracleSemModel(
      oracle, szModelName);

    model.getGraph().add(Triple.create(
          Node.createURI("http://example.com/John"),
          Node.createURI("http://example.com/fatherOf"),
          Node.createURI("http://example.com/Mary")));
    Query query = QueryFactory.create(
        "select ?f ?k WHERE {?f <http://example.com/fatherOf> ?k .}");
    QueryExecution qexec = QueryExecutionFactory.create(query, model);
    ResultSet results = qexec.execSelect();
    ResultSetFormatter.out(System.out, results, query);
    model.close();
    oracle.dispose();
  }
}
```

The following are the commands to compile and run Example D-1, as well as the expected output of the `java` command.

```
javac -classpath ../jar/'*' Test.java
java -classpath ./:../jar/'*'  Test jdbc:oracle:thin:@localhost:1521:orcl scott
<password-for-scott> M1

---------------------------------------------------------
| f                         | k                         |
=========================================================
| <http://example.com/John> | <http://example.com/Mary> |
---------------------------------------------------------
```

D.3.2 Test6.java: Load OWL Ontology and Perform OWLPrime inference

Example D-2 loads an OWL ontology and performs OWLPrime inference. Note that the OWL ontology is in RDF/XML format, and after it is loaded into Oracle it will be serialized out in N-TRIPLE form. The example also queries for the number of asserted and inferred triples.

The ontology in this example can be retrieved from `http://swat.cse.lehigh.edu/onto/univ-bench.owl`, and it describes roles, resources, and relationships in a university environment.

Example D-2 Load OWL Ontology and Perform OWLPrime inference

```
import java.io.*;
import org.apache.jena.query.*;
import org.apache.jena.rdf.model.Model;
import org.apache.jena.util.FileManager;
import oracle.spatial.rdf.client.jena.*;
public class Test6 {
  public static void main(String[] args) throws Exception
  {
    String szJdbcURL = args[0];
    String szUser    = args[1];
    String szPasswd  = args[2];
    String szModelName = args[3];

    Oracle oracle = new Oracle(szJdbcURL, szUser, szPasswd);
    Model model = ModelOracleSem.createOracleSemModel(oracle, szModelName);

    // load UNIV ontology
    InputStream in = FileManager.get().open("./univ-bench.owl" );
    model.read(in, null);
    OutputStream os = new FileOutputStream("./univ-bench.nt");
    model.write(os, "N-TRIPLE");
    os.close();

    String queryString =
      " SELECT ?subject ?prop ?object WHERE { ?subject ?prop ?object } ";

    Query query = QueryFactory.create(queryString) ;
    QueryExecution qexec = QueryExecutionFactory.create(query, model) ;

    try {
      int iTriplesCount = 0;
      ResultSet results = qexec.execSelect() ;
      for ( ; results.hasNext() ; ) {
        QuerySolution soln = results.nextSolution() ;
        iTriplesCount++;
      }
      System.out.println("Asserted  triples count: " + iTriplesCount);
    }
    finally {
      qexec.close() ;
    }

    Attachment attachment = Attachment.createInstance(
        new String[] {}, "OWLPRIME",
        InferenceMaintenanceMode.NO_UPDATE, QueryOptions.DEFAULT);

    GraphOracleSem graph = new GraphOracleSem(oracle, szModelName, attachment);
    graph.analyze();
    graph.performInference();

    query = QueryFactory.create(queryString) ;
    qexec = QueryExecutionFactory.create(query,new ModelOracleSem(graph)) ;

    try {
      int iTriplesCount = 0;
      ResultSet results = qexec.execSelect() ;
      for ( ; results.hasNext() ; ) {
        QuerySolution soln = results.nextSolution() ;
        iTriplesCount++;
      }
```

```
      System.out.println("Asserted + Infered triples count: " + iTriplesCount);
    }
    finally {
      qexec.close() ;
    }
    model.close();

    OracleUtils.dropSemanticModel(oracle, szModelName);
    oracle.dispose();
  }
}
```

The following are the commands to compile and run Example D-2, as well as the expected output of the `java` command.

```
javac -classpath ../jar/'*' Test6.java
java -classpath ./:../jar/'*'  Test6 jdbc:oracle:thin:@localhost:1521:orcl scott
<password-for-scott> M1
Asserted  triples count: 293
Asserted + Infered triples count: 340
```

Note that this output reflects an older version of the LUBM ontology. The latest version of the ontology has more triples.

D.3.3 Test7.java: Bulk Load OWL Ontology and Perform OWLPrime inference

Example D-3 loads the same OWL ontology as in Test6.java: Load OWL Ontology and Perform OWLPrime inference, but stored in a local file using Bulk Loader. Ontologies can also be loaded using an incremental and batch loader; these two methods are also listed in the example for completeness.

Example D-3 Bulk Load OWL Ontology and Perform OWLPrime inference

```
import java.io.*;
import org.apache.jena.graph.*;
import org.apache.jena.rdf.model.*;
import org.apache.jena.util.*;
import oracle.spatial.rdf.client.jena.*;

public class Test7
{
  public static void main(String[] args) throws Exception
  {
    String szJdbcURL = args[0];
    String szUser    = args[1];
    String szPasswd  = args[2];
    String szModelName = args[3];
    // in memory Jena Model
    Model model = ModelFactory.createDefaultModel();
    InputStream is = FileManager.get().open("./univ-bench.owl");
    model.read(is, "", "RDF/XML");
    is.close();

    Oracle oracle = new Oracle(szJdbcURL, szUser, szPasswd);
    ModelOracleSem modelDest = ModelOracleSem.createOracleSemModel(oracle,
szModelName);

    GraphOracleSem g = modelDest.getGraph();
    g.dropApplicationTableIndex();
```

```
    int method = 2; // try bulk loader
    String tbs = "SYSAUX"; // can be customized
    if (method == 0) {
      System.out.println("start incremental");
      modelDest.add(model);
      System.out.println("end size " + modelDest.size());
    }
    else if (method == 1) {
      System.out.println("start batch load");
      g.getBulkUpdateHandler().addInBatch(
          GraphUtil.findAll(model.getGraph()), tbs);
      System.out.println("end size " + modelDest.size());
    }
    else {
      System.out.println("start bulk load");
      g.getBulkUpdateHandler().addInBulk(
          GraphUtil.findAll(model.getGraph()), tbs);
      System.out.println("end size " + modelDest.size());
    }
    g.rebuildApplicationTableIndex();

    long lCount = g.getCount(Triple.ANY);
    System.out.println("Asserted  triples count: " + lCount);
    model.close();
    OracleUtils.dropSemanticModel(oracle, szModelName);
    oracle.dispose();
  }
}
```

The following are the commands to compile and run Example D-3, as well as the expected output of the `java` command.

```
javac -classpath ../jar/'*' Test7.java
java -classpath ./:../jar/'*'  Test7 jdbc:oracle:thin:@localhost:1521:orcl scott
<password-for-scott> M1
start bulk load
end size 293
Asserted  triples count: 293
```

Note that this output reflects an older version of the LUBM ontology. The latest version of the ontology has more triples.

D.3.4 Test8.java: SPARQL OPTIONAL Query

Example D-4 shows a SPARQL OPTIONAL query. It inserts triples that postulate the following:

- John is a parent of Mary.
- John is a parent of Jack.
- Mary is a parent of Jill.

It then finds parent-child relationships, optionally including any grandchild (gkid) relationships.

Example D-4 SPARQL OPTIONAL Query

```
import org.apache.jena.query.*;
import oracle.spatial.rdf.client.jena.*;
import org.apache.jena.graph.*;

public class Test8
{
```

```
  public static void main(String[] args) throws Exception
  {
    String szJdbcURL = args[0];
    String szUser    = args[1];
    String szPasswd  = args[2];
    String szModelName = args[3];

    Oracle oracle = new Oracle(szJdbcURL, szUser, szPasswd);
    ModelOracleSem model = ModelOracleSem.createOracleSemModel(oracle, 
szModelName);
    GraphOracleSem g = model.getGraph();

    g.add(Triple.create(
          Node.createURI("u:John"), Node.createURI("u:parentOf"), 
Node.createURI("u:Mary")));
    g.add(Triple.create(
          Node.createURI("u:John"), Node.createURI("u:parentOf"), 
Node.createURI("u:Jack")));
    g.add(Triple.create(
          Node.createURI("u:Mary"), Node.createURI("u:parentOf"), 
Node.createURI("u:Jill")));

    String queryString =
  " SELECT ?s ?o ?gkid " +
  " WHERE { ?s <u:parentOf> ?o . OPTIONAL {?o <u:parentOf> ?gkid }} ";

    Query query = QueryFactory.create(queryString) ;
    QueryExecution qexec = QueryExecutionFactory.create(query, model) ;

    try {
      int iMatchCount = 0;
      ResultSet results = qexec.execSelect() ;
      ResultSetFormatter.out(System.out, results, query);
    }
    finally {
      qexec.close() ;
    }
    model.close();

    OracleUtils.dropSemanticModel(oracle, szModelName);
    oracle.dispose();
  }
}
```

The following are the commands to compile and run Example D-4, as well as the expected output of the `java` command.

```
javac -classpath ../jar/'*' Test8.java
java -classpath ./:../jar/'*'  Test8 jdbc:oracle:thin:@localhost:1521:orcl scott 
<password-for-scott> M1
----------------------------------
| s        | o        | gkid     |
==================================
| <u:John> | <u:Mary> | <u:Jill> |
| <u:Mary> | <u:Jill> |          |
| <u:John> | <u:Jack> |          |
----------------------------------
```

D.3.5 Test9.java: SPARQL Query with LIMIT and OFFSET

Example D-5 shows a SPARQL query with LIMIT and OFFSET. It inserts triples that postulate the following:

- John is a parent of Mary.
- John is a parent of Jack.
- Mary is a parent of Jill.

It then finds one parent-child relationship (LIMIT 1), skipping the first two parent-child relationships encountered (OFFSET 2), and optionally includes any grandchild (gkid) relationships for the one found.

Example D-5   SPARQL Query with LIMIT and OFFSET

```
import org.apache.jena.query.*;
import oracle.spatial.rdf.client.jena.*;
import org.apache.jena.graph.*;
public class Test9
{
  public static void main(String[] args) throws Exception
  {
    String szJdbcURL = args[0];
    String szUser    = args[1];
    String szPasswd  = args[2];
    String szModelName = args[3];

    Oracle oracle = new Oracle(szJdbcURL, szUser, szPasswd);
    ModelOracleSem model = ModelOracleSem.createOracleSemModel(oracle, 
szModelName);
    GraphOracleSem g = model.getGraph();

    g.add(Triple.create(Node.createURI("u:John"), Node.createURI("u:parentOf"),
                    Node.createURI("u:Mary")));
    g.add(Triple.create(Node.createURI("u:John"), Node.createURI("u:parentOf"),
                    Node.createURI("u:Jack")));
    g.add(Triple.create(Node.createURI("u:Mary"), 
Node.createURI("u:parentOf"),
                    Node.createURI("u:Jill")));

    String queryString =
      " SELECT ?s ?o ?gkid " +
      " WHERE { ?s <u:parentOf> ?o . OPTIONAL {?o <u:parentOf> ?gkid }} " +
      " LIMIT 1 OFFSET 2";

    Query query = QueryFactory.create(queryString) ;
    QueryExecution qexec = QueryExecutionFactory.create(query, model) ;

    int iMatchCount = 0;
    ResultSet results = qexec.execSelect() ;
    ResultSetFormatter.out(System.out, results, query);
    qexec.close() ;
    model.close();

    OracleUtils.dropSemanticModel(oracle, szModelName);
    oracle.dispose();
  }
}
```

The following are the commands to compile and run Example D-5, as well as the expected output of the `java` command.

```
javac -classpath ../jar/'*' Test9.java
java -classpath ./:../jar/'*'  Test9 jdbc:oracle:thin:@localhost:1521:orcl scott
<password-for-scott> M1
------------------------------
| s        | o        | gkid |
==============================
| <u:John> | <u:Jack> |      |
------------------------------
```

D.3.6 Test10.java: SPARQL Query with TIMEOUT and DOP

Example D-6 shows the SPARQL query from Test9.java: SPARQL Query with LIMIT and OFFSET with additional features, including a timeout setting (TIMEOUT=1, in seconds) and parallel execution setting (DOP=4).

Example D-6   SPARQL Query with TIMEOUT and DOP

```
import org.apache.jena.query.*;
import oracle.spatial.rdf.client.jena.*;
import org.apache.jena.graph.*;

public class Test10 {
  public static void main(String[] args) throws Exception  {
    String szJdbcURL = args[0];
    String szUser    = args[1];
    String szPasswd  = args[2];
    String szModelName = args[3];

    Oracle oracle = new Oracle(szJdbcURL, szUser, szPasswd);
    ModelOracleSem model = ModelOracleSem.createOracleSemModel(oracle, szModelName);
    GraphOracleSem g = model.getGraph();

    g.add(Triple.create(Node.createURI("u:John"), Node.createURI("u:parentOf"),
                        Node.createURI("u:Mary")));
    g.add(Triple.create(Node.createURI("u:John"), Node.createURI("u:parentOf"),
                    Node.createURI("u:Jack")));
    g.add(Triple.create(Node.createURI("u:Mary"), Node.createURI("u:parentOf"),
                    Node.createURI("u:Jill")));
    String queryString =
        " PREFIX ORACLE_SEM_FS_NS: <http://oracle.com/semtech#dop=4,timeout=1> "
      + " SELECT ?s ?o ?gkid WHERE { ?s <u:parentOf> ?o . "
      + " OPTIONAL {?o <u:parentOf> ?gkid }} "
      + " LIMIT 1 OFFSET 2";

    Query query = QueryFactory.create(queryString) ;
    QueryExecution qexec = QueryExecutionFactory.create(query, model) ;

    int iMatchCount = 0;
    ResultSet results = qexec.execSelect() ;
    ResultSetFormatter.out(System.out, results, query);
    qexec.close() ;
    model.close();

    OracleUtils.dropSemanticModel(oracle, szModelName);
    oracle.dispose();
  }
}
```

The following are the commands to compile and run Example D-6, as well as the expected output of the `java` command.

```
javac -classpath ../jar/'*' Test10.java
java -classpath ./:../jar/'*'  Test10 jdbc:oracle:thin:@localhost:1521:orcl scott
<password-for-scott> M1
------------------------------
| s         | o          | gkid |
==============================
| <u:John> | <u:Jack> |      |
------------------------------
```

D.3.7 Test11.java: Query Involving Named Graphs

Example D-7 shows a query involving named graphs. It involves a default graph that has information about named graph URIs and their publishers. The query finds graph names, their publishers, and within each named graph finds the mailbox value using the `foaf:mbox` predicate.

Example D-7 Named Graph Based Query

```
import org.apache.jena.query.*;
import oracle.spatial.rdf.client.jena.*;
import org.apache.jena.graph.*;

public class Test11
{
  public static void main(String[] args) throws Exception
  {
    String szJdbcURL = args[0];
    String szUser    = args[1];
    String szPasswd  = args[2];
    String szModelName = args[3];

    Oracle oracle = new Oracle(szJdbcURL, szUser, szPasswd);
    GraphOracleSem graph = new GraphOracleSem(oracle, szModelName);
    DatasetGraphOracleSem dataset = DatasetGraphOracleSem.createFrom(graph);

    // don't need the GraphOracleSem anymore, release resources
    graph.close();

    // add data to the default graph
    dataset.add(new Quad(
          Quad.defaultGraphIRI, // specifies default graph
          Node.createURI("http://example.org/bob"),
          Node.createURI("http://purl.org/dc/elements/1.1/publisher"),
          Node.createLiteral("Bob Hacker")));
    dataset.add(new Quad(
          Quad.defaultGraphIRI, // specifies default graph
          Node.createURI("http://example.org/alice"),
          Node.createURI("http://purl.org/dc/elements/1.1/publisher"),
          Node.createLiteral("alice Hacker")));

    // add data to the bob named graph
    dataset.add(new Quad(
          Node.createURI("http://example.org/bob"), // graph name
          Node.createURI("urn:bob"),
          Node.createURI("http://xmlns.com/foaf/0.1/name"),
          Node.createLiteral("Bob")));
    dataset.add(new Quad(
          Node.createURI("http://example.org/bob"), // graph name
```

```
        Node.createURI("urn:bob"),
        Node.createURI("http://xmlns.com/foaf/0.1/mbox"),
        Node.createURI("mailto:bob@example")));

  // add data to the alice named graph
  dataset.add(new Quad(
        Node.createURI("http://example.org/alice"), // graph name
        Node.createURI("urn:alice"),
        Node.createURI("http://xmlns.com/foaf/0.1/name"),
        Node.createLiteral("Alice")));
  dataset.add(new Quad(
        Node.createURI("http://example.org/alice"), // graph name
        Node.createURI("urn:alice"),
        Node.createURI("http://xmlns.com/foaf/0.1/mbox"),
        Node.createURI("mailto:alice@example")));

  DataSource ds = DatasetFactory.create(dataset);

  String queryString =
        " PREFIX foaf: <http://xmlns.com/foaf/0.1/> "
      + " PREFIX dc: <http://purl.org/dc/elements/1.1/> "
      + " SELECT ?who ?graph ?mbox "
      + " FROM NAMED <http://example.org/alice> "
      + " FROM NAMED <http://example.org/bob> "
      + " WHERE "
      + " { "
      + "    ?graph dc:publisher ?who . "
      + "    GRAPH ?graph { ?x foaf:mbox ?mbox } "
      + " } ";

  Query query = QueryFactory.create(queryString);
  QueryExecution qexec = QueryExecutionFactory.create(query, ds);

  ResultSet results = qexec.execSelect();
  ResultSetFormatter.out(System.out, results, query);

  qexec.close();
  dataset.close();

  oracle.dispose();
 }
}
```

The following are the commands to compile and run Example D-7, as well as the expected output of the `java` command.

```
javac -classpath ./:./jena-2.6.4.jar:./sdordfclient.jar:./ojdbc6.jar:./slf4j-
api-1.5.8.jar:./slf4j-log4j12-1.5.8.jar:./arq-2.8.8.jar:./xercesImpl-2.7.1.jar
Test11.java
java -classpath ./:../jar/'*'  Test11 jdbc:oracle:thin:@localhost:1521:orcl scott
<password-for-scott> M1
------------------------------------------------------------------------
| who            | graph                        | mbox                  |
========================================================================
| "alice Hacker" | <http://example.org/alice> | <mailto:alice@example> |
| "Bob Hacker"   | <http://example.org/bob>   | <mailto:bob@example>   |
------------------------------------------------------------------------
```

D.3.8 Test12.java: SPARQL ASK Query

Example D-8 shows a SPARQL ASK query. It inserts a triple that postulates that John is a parent of Mary. It then finds whether John is a parent of Mary.

Example D-8   SPARQL ASK Query

```
import org.apache.jena.query.*;
import oracle.spatial.rdf.client.jena.*;
import org.apache.jena.graph.*;
{
  public static void main(String[] args) throws Exception
  {
    String szJdbcURL = args[0];
    String szUser    = args[1];
    String szPasswd  = args[2];
    String szModelName = args[3];

    Oracle oracle = new Oracle(szJdbcURL, szUser, szPasswd);
    ModelOracleSem model = ModelOracleSem.createOracleSemModel(oracle,
          szModelName);
    GraphOracleSem g = model.getGraph();

    g.add(Triple.create(Node.createURI("u:John"), Node.createURI("u:parentOf"),
                        Node.createURI("u:Mary")));
    String queryString = " ASK { <u:John> <u:parentOf> <u:Mary> } ";

    Query query = QueryFactory.create(queryString) ;
    QueryExecution qexec = QueryExecutionFactory.create(query, model) ;
    boolean b = qexec.execAsk();
    System.out.println("ask result = " + ((b)?"TRUE":"FALSE"));
    qexec.close() ;

    model.close();
    OracleUtils.dropSemanticModel(oracle, szModelName);
    oracle.dispose();
  }
}
```

The following are the commands to compile and run Example D-8, as well as the expected output of the `java` command.

```
javac -classpath ../jar/'*' Test12.java
java -classpath ./:../jar/'*'  Test12 jdbc:oracle:thin:@localhost:1521:orcl scott
<password-for-scott> M1
ask result = TRUE
```

D.3.9 Test13.java: SPARQL DESCRIBE Query

Example D-9 shows a SPARQL DESCRIBE query. It inserts triples that postulate the following:

- John is a parent of Mary.
- John is a parent of Jack.
- Amy is a parent of Jack.

It then finds all relationships that involve any parents of Jack.

Example D-9 SPARQL DESCRIBE Query

```
import org.apache.jena.query.*;
import org.apache.jena.rdf.model.Model;
import oracle.spatial.rdf.client.jena.*;
import org.apache.jena.graph.*;

public class Test13
{
  public static void main(String[] args) throws Exception
  {
    String szJdbcURL = args[0];
    String szUser    = args[1];
    String szPasswd  = args[2];
    String szModelName = args[3];

    Oracle oracle = new Oracle(szJdbcURL, szUser, szPasswd);
    ModelOracleSem model = ModelOracleSem.createOracleSemModel(oracle, szModelName);
    GraphOracleSem g = model.getGraph();

    g.add(Triple.create(Node.createURI("u:John"), Node.createURI("u:parentOf"),
                Node.createURI("u:Mary")));
    g.add(Triple.create(Node.createURI("u:John"), Node.createURI("u:parentOf"),
 Node.createURI("u:Jack")));
    g.add(Triple.create(Node.createURI("u:Amy"), Node.createURI("u:parentOf"),
 Node.createURI("u:Jack")));
    String queryString = " DESCRIBE ?x WHERE {?x <u:parentOf> <u:Jack>}";

    Query query = QueryFactory.create(queryString) ;
    QueryExecution qexec = QueryExecutionFactory.create(query, model) ;
    Model m = qexec.execDescribe();
    System.out.println("describe result = " + m.toString());

    qexec.close() ;
    model.close();
    OracleUtils.dropSemanticModel(oracle, szModelName);
    oracle.dispose();
  }
}
```

The following are the commands to compile and run Example D-9, as well as the expected output of the `java` command.

```
javac -classpath ../jar/'*' Test13.java
java -classpath ./:../jar/'*'  Test13 jdbc:oracle:thin:@localhost:1521:orcl scott
<password-for-scott> M1
describe result = <ModelCom   {u:Amy @u:parentOf u:Jack;
     u:John @u:parentOf u:Jack; u:John @u:parentOf u:Mary} |  [u:Amy, u:parentOf,
u:Jack] [u:John, u:parentOf,
       u:Jack] [u:John, u:parentOf, u:Mary]>
```

D.3.10 Test14.java: SPARQL CONSTRUCT Query

Example D-10 shows a SPARQL CONSTRUCT query. It inserts triples that postulate the following:

- John is a parent of Mary.
- John is a parent of Jack.
- Amy is a parent of Jack.
- Each parent loves their children.

It then constructs an RDF graph with information about who loves whom.

Example D-10 SPARQL CONSTRUCT Query

```
import org.apache.jena.query.*;
import org.apache.jena.rdf.model.Model;
import oracle.spatial.rdf.client.jena.*;
import org.apache.jena.graph.*;

public class Test14
{
  public static void main(String[] args) throws Exception
  {
    String szJdbcURL = args[0];
    String szUser    = args[1];
    String szPasswd  = args[2];
    String szModelName = args[3];

    Oracle oracle = new Oracle(szJdbcURL, szUser, szPasswd);
    ModelOracleSem model = ModelOracleSem.createOracleSemModel(oracle, szModelName);
    GraphOracleSem g = model.getGraph();

    g.add(Triple.create(Node.createURI("u:John"), Node.createURI("u:parentOf"),
 Node.createURI("u:Mary")));
    g.add(Triple.create(Node.createURI("u:John"), Node.createURI("u:parentOf"),
 Node.createURI("u:Jack")));
    g.add(Triple.create(Node.createURI("u:Amy"), Node.createURI("u:parentOf"),
 Node.createURI("u:Jack")));
    String queryString = " CONSTRUCT { ?s <u:loves> ?o } WHERE {?s <u:parentOf> ?o}";

    Query query = QueryFactory.create(queryString) ;
    QueryExecution qexec = QueryExecutionFactory.create(query, model) ;
    Model m = qexec.execConstruct();
    System.out.println("Construct result = " + m.toString());

    qexec.close() ;
    model.close();
    OracleUtils.dropSemanticModel(oracle, szModelName);
    oracle.dispose();
  }
}
```

The following are the commands to compile and run Example D-10, as well as the expected output of the `java` command.

```
javac -classpath ../jar/'*' Test14.java
java -classpath ./:../jar/'*'  Test14 jdbc:oracle:thin:@localhost:1521:orcl scott
<password-for-scott> M1
Construct result = <ModelCom   {u:Amy @u:loves u:Jack;
  u:John @u:loves u:Jack; u:John @u:loves u:Mary} |  [u:Amy, u:loves, u:Jack] [u:John,
u:loves,
    u:Jack] [u:John, u:loves, u:Mary]>
```

D.3.11 Test15.java: Query Multiple Models and Specify "Allow Duplicates"

Example D-11 queries multiple models and uses the "allow duplicates" option. It inserts triples that postulate the following:

- John is a parent of Jack (in Model 1).
- Mary is a parent of Jack (in Model 2).

- Each parent loves their children.

It then finds out who loves whom. It searches both models and allows for the possibility of duplicate triples in the models (although there are no duplicates in this example).

Example D-11 Query Multiple Models and Specify "Allow Duplicates"

```
import org.apache.jena.query.*;
import org.apache.jena.rdf.model.Model;
import oracle.spatial.rdf.client.jena.*;
import org.apache.jena.graph.*;

public class Test15
{
  public static void main(String[] args) throws Exception
  {
    String szJdbcURL = args[0];
    String szUser    = args[1];
    String szPasswd  = args[2];
    String szModelName1 = args[3];
    String szModelName2 = args[4];

    Oracle oracle = new Oracle(szJdbcURL, szUser, szPasswd);
    ModelOracleSem model1 = ModelOracleSem.createOracleSemModel(oracle, szModelName1);
    model1.getGraph().add(Triple.create(Node.createURI("u:John"),
                     Node.createURI("u:parentOf"), Node.createURI("u:Jack")));
    model1.close();

    ModelOracleSem model2 = ModelOracleSem.createOracleSemModel(oracle, szModelName2);
    model2.getGraph().add(Triple.create(Node.createURI("u:Mary"),
                     Node.createURI("u:parentOf"), Node.createURI("u:Jack")));
    model2.close();

    String[] modelNamesList = {szModelName2};
    String[] rulebasesList  = {};
    Attachment attachment = Attachment.createInstance(modelNamesList, rulebasesList,
              InferenceMaintenanceMode.NO_UPDATE,
              QueryOptions.ALLOW_QUERY_VALID_AND_DUP);

    GraphOracleSem graph = new GraphOracleSem(oracle, szModelName1, attachment);
    ModelOracleSem model = new ModelOracleSem(graph);

    String queryString = " CONSTRUCT { ?s <u:loves> ?o } WHERE {?s <u:parentOf> ?o}";
    Query query = QueryFactory.create(queryString) ;
    QueryExecution qexec = QueryExecutionFactory.create(query, model) ;
    Model m = qexec.execConstruct();
    System.out.println("Construct result = " + m.toString());

    qexec.close() ;
    model.close();
    OracleUtils.dropSemanticModel(oracle, szModelName1);
    OracleUtils.dropSemanticModel(oracle, szModelName2);
    oracle.dispose();
  }
}
```

The following are the commands to compile and run Example D-11, as well as the expected output of the `java` command.

```
javac -classpath ../jar/'*' Test15.java
java -classpath ./:../jar/'*'  Test15 jdbc:oracle:thin:@localhost:1521:orcl scott
<password-for-scott> M1 M2
```

```
Construct result = <ModelCom   {u:Mary @u:loves u:Jack; u:John @u:loves u:Jack} |
[u:Mary, u:loves, u:Jack] [u:John, u:loves, u:Jack]>
```

D.3.12 Test16.java: SPARQL Update

Example D-12 inserts two triples into a model.

Example D-12 SPARQL Update

```
import org.apache.jena.util.iterator.*;
import oracle.spatial.rdf.client.jena.*;
import org.apache.jena.graph.*;
import org.apache.jena.update.*;

public class Test16
{
  public static void main(String[] args) throws Exception
  {
    String szJdbcURL = args[0];
    String szUser    = args[1];
    String szPasswd  = args[2];
    String szModelName = args[3];

    Oracle oracle = new Oracle(szJdbcURL, szUser, szPasswd);
    ModelOracleSem model = ModelOracleSem.createOracleSemModel(oracle, szModelName);
    GraphOracleSem g = model.getGraph();
    String insertString =
      " PREFIX dc: <http://purl.org/dc/elements/1.1/> "         +
      " INSERT DATA "                                           +
      " { <http://example/book3> dc:title    \"A new book\" ; " +
      "                          dc:creator  \"A.N.Other\" . "   +
      " }   ";

    UpdateAction.parseExecute(insertString,  model);
    ExtendedIterator ei = GraphUtil.findAll(g);
    while (ei.hasNext()) {
      System.out.println("Triple " + ei.next().toString());
    }
    model.close();
    OracleUtils.dropSemanticModel(oracle, szModelName);
    oracle.dispose();
  }
}
```

The following are the commands to compile and run Example D-12, as well as the expected output of the `java` command.

```
javac -classpath ../jar/'*' Test16.java
java -classpath ./:../jar/'*'  Test16 jdbc:oracle:thin:@localhost:1521:orcl scott
<password-for-scott> M1
Triple http://example/book3 @dc:title "A new book"
Triple http://example/book3 @dc:creator "A.N.Other"
```

D.3.13 Test17.java: SPARQL Query with ARQ Built-In Functions

Example D-13 inserts data about two books, and it displays the book titles in all uppercase characters and the length of each title string.

Example D-13   SPARQL Query with ARQ Built-In Functions

```
import org.apache.jena.query.*;
import oracle.spatial.rdf.client.jena.*;
import org.apache.jena.update.*;

public class Test17 {
  public static void main(String[] args) throws Exception  {
    String szJdbcURL = args[0];
    String szUser    = args[1];
    String szPasswd  = args[2];
    String szModelName = args[3];

    Oracle oracle = new Oracle(szJdbcURL, szUser, szPasswd);
    ModelOracleSem model = ModelOracleSem.createOracleSemModel(oracle, szModelName);
    GraphOracleSem g = model.getGraph();
    String insertString =
      " PREFIX dc: <http://purl.org/dc/elements/1.1/> "         +
      " INSERT DATA "                                           +
      " { <http://example/book3> dc:title    \"A new book\" ; " +
      "                          dc:creator  \"A.N.Other\" . "  +
      "   <http://example/book4> dc:title    \"Semantic Web Rocks\" ; " +
      "                          dc:creator  \"TB\" . "   +
      " }   ";

    UpdateAction.parseExecute(insertString,  model);
    String queryString = "PREFIX  dc:   <http://purl.org/dc/elements/1.1/> " +
      " PREFIX  fn: <http://www.w3.org/2005/xpath-functions#> " +
      " SELECT ?subject (fn:upper-case(?object) as ?object1)  " +
      "                 (fn:string-length(?object) as ?strlen) " +
      " WHERE { ?subject dc:title ?object } "
      ;
    Query query = QueryFactory.create(queryString, Syntax.syntaxARQ);
    QueryExecution qexec = QueryExecutionFactory.create(query, model);
    ResultSet results = qexec.execSelect();
    ResultSetFormatter.out(System.out, results, query);
    model.close();
    OracleUtils.dropSemanticModel(oracle, szModelName);
    oracle.dispose();
  }
}
```

The following are the commands to compile and run Example D-13, as well as the expected output of the `java` command.

```
javac -classpath ../jar/'*' Test17.java
java -classpath ./:../jar/'*'  Test17 jdbc:oracle:thin:@localhost:1521:orcl scott
<password-for-scott> M1
----------------------------------------------------------
| subject                | object1               | strlen |
==========================================================
| <http://example/book3> | "A NEW BOOK"          | 10     |
| <http://example/book4> | "SEMANTIC WEB ROCKS" | 18     |
----------------------------------------------------------
```

D.3.14 Test18.java: SELECT Cast Query

Example D-14 "converts" two Fahrenheit temperatures (18.1 and 32.0) to Celsius temperatures.

Example D-14   SELECT Cast Query

```
import org.apache.jena.query.*;
import oracle.spatial.rdf.client.jena.*;
import org.apache.jena.update.*;

public class Test18 {
  public static void main(String[] args) throws Exception  {
    String szJdbcURL = args[0];
    String szUser    = args[1];
    String szPasswd  = args[2];
    String szModelName = args[3];

    Oracle oracle = new Oracle(szJdbcURL, szUser, szPasswd);
    ModelOracleSem model = ModelOracleSem.createOracleSemModel(oracle, 
szModelName);
    GraphOracleSem g = model.getGraph();
    String insertString =
      " PREFIX xsd: <http://www.w3.org/2001/XMLSchema#> " +
      " INSERT DATA "                                     +
      " { <u:Object1> <u:temp>    \"18.1\"^^xsd:float ; " +
      "               <u:name>    \"Foo... \" . "         +
      "   <u:Object2> <u:temp>    \"32.0\"^^xsd:float ; " +
      "               <u:name>    \"Bar... \" . "         +
      " }   ";

    UpdateAction.parseExecute(insertString,  model);
    String queryString =
      " PREFIX  fn: <http://www.w3.org/2005/xpath-functions#> " +
      " SELECT ?subject ((?temp - 32.0)*5/9 as ?celsius_temp) " +
      "WHERE { ?subject <u:temp> ?temp } "
      ;
    Query query = QueryFactory.create(queryString, Syntax.syntaxARQ);
    QueryExecution qexec = QueryExecutionFactory.create(query, model);
    ResultSet results = qexec.execSelect();
    ResultSetFormatter.out(System.out, results, query);

    model.close();
    OracleUtils.dropSemanticModel(oracle, szModelName);
    oracle.dispose();
  }
}
```

The following are the commands to compile and run Example D-14, as well as the expected output of the `java` command.

```
javac -classpath ../jar/'*' Test18.java
java -classpath ./:../jar/'*'  Test18 jdbc:oracle:thin:@localhost:1521:orcl scott 
<password-for-scott> M1
------------------------------------------------------------------------
| subject     | celsius_temp                                           |
========================================================================
| <u:Object1> | "-7.7222223"^^<http://www.w3.org/2001/XMLSchema#float> |
| <u:Object2> | "0.0"^^<http://www.w3.org/2001/XMLSchema#float>        |
------------------------------------------------------------------------
```

D.3.15 Test19.java: Instantiate Oracle Database Using OracleConnection

Example D-15 shows a different way to instantiate an Oracle object using a given `OracleConnection` object. (In a J2EE Web application, users can normally get an `OracleConnection` object from a J2EE data source.)

Example D-15   Instantiate Oracle Database Using OracleConnection

```
import org.apache.jena.query.*;
import org.apache.jena.graph.*;
import oracle.spatial.rdf.client.jena.*;
import oracle.jdbc.pool.*;
import oracle.jdbc.*;

public class Test19 {
  public static void main(String[] args) throws Exception {
    String szJdbcURL = args[0];
    String szUser    = args[1];
    String szPasswd  = args[2];
    String szModelName = args[3];

    OracleDataSource ds = new OracleDataSource();
    ds.setURL(szJdbcURL);
    ds.setUser(szUser);
    ds.setPassword(szPasswd);
    OracleConnection conn = (OracleConnection) ds.getConnection();
    Oracle oracle = new Oracle(conn);

    ModelOracleSem model = ModelOracleSem.createOracleSemModel(oracle,
szModelName);
    GraphOracleSem g = model.getGraph();

    g.add(Triple.create(Node.createURI("u:John"), Node.createURI("u:parentOf"),
                        Node.createURI("u:Mary")));
    g.add(Triple.create(Node.createURI("u:John"), Node.createURI("u:parentOf"),
                        Node.createURI("u:Jack")));
    g.add(Triple.create(Node.createURI("u:Mary"), Node.createURI("u:parentOf"),
         Node.createURI("u:Jill")));
    String queryString =
       " SELECT ?s ?o  WHERE { ?s <u:parentOf> ?o .} ";
    Query query = QueryFactory.create(queryString) ;
    QueryExecution qexec = QueryExecutionFactory.create(query, model) ;

    ResultSet results = qexec.execSelect() ;
    ResultSetFormatter.out(System.out, results, query);
    qexec.close() ;
    model.close();
    OracleUtils.dropSemanticModel(oracle, szModelName);
    oracle.dispose();
  }
}
```

The following are the commands to compile and run Example D-15, as well as the expected output of the `java` command.

```
javac -classpath ../jar/'*' Test19.java
java -classpath ./:../jar/'*'  Test19 jdbc:oracle:thin:@localhost:1521:orcl scott
<password-for-scott> M1
-----------------------
| s        | o        |
=======================
| <u:John> | <u:Mary> |
| <u:John> | <u:Jack> |
| <u:Mary> | <u:Jill> |
-----------------------
```

D.3.16 Test20.java: Oracle Database Connection Pooling

Example D-16 uses Oracle Database connection pooling.

Example D-16 Oracle Database Connection Pooling

```
import org.apache.jena.graph.*;
import oracle.spatial.rdf.client.jena.*;

public class Test20
{
  public static void main(String[] args) throws Exception
  {
    String szJdbcURL = args[0];
    String szUser    = args[1];
    String szPasswd  = args[2];
    String szModelName = args[3];

    // test with connection properties (taken from some example)
    java.util.Properties prop = new java.util.Properties();
    prop.setProperty("MinLimit", "2");     // the cache size is 2 at least
    prop.setProperty("MaxLimit", "10");
    prop.setProperty("InitialLimit", "2"); // create 2 connections at startup
    prop.setProperty("InactivityTimeout", "1800");    //  seconds
    prop.setProperty("AbandonedConnectionTimeout", "900");  //  seconds
    prop.setProperty("MaxStatementsLimit", "10");
    prop.setProperty("PropertyCheckInterval", "60"); // seconds

    System.out.println("Creating OraclePool");
    OraclePool op = new OraclePool(szJdbcURL, szUser, szPasswd, prop,
               "OracleSemConnPool");
    System.out.println("Done creating OraclePool");

    // grab an Oracle and do something with it
    System.out.println("Getting an Oracle from OraclePool");
    Oracle oracle = op.getOracle();
    System.out.println("Done");
    System.out.println("Is logical connection:" +
        oracle.getConnection().isLogicalConnection());
    GraphOracleSem g = new GraphOracleSem(oracle, szModelName);
    g.add(Triple.create(Node.createURI("u:John"), Node.createURI("u:parentOf"),
                        Node.createURI("u:Mary")));
    g.close();
    // return the Oracle back to the pool
    oracle.dispose();

    // grab another Oracle and do something else
    System.out.println("Getting an Oracle from OraclePool");
    oracle = op.getOracle();
    System.out.println("Done");
    System.out.println("Is logical connection:" +
        oracle.getConnection().isLogicalConnection());
    g = new GraphOracleSem(oracle, szModelName);
    g.add(Triple.create(Node.createURI("u:John"), Node.createURI("u:parentOf"),
                        Node.createURI("u:Jack")));
    g.close();

    OracleUtils.dropSemanticModel(oracle, szModelName);

    // return the Oracle back to the pool
    oracle.dispose();
```

```
  }
}
```

The following are the commands to compile and run Example D-16, as well as the expected output of the `java` command.

```
javac -classpath ../jar/'*' Test20.java
java -classpath ./:../jar/'*'  Test20 jdbc:oracle:thin:@localhost:1521:orcl scott
<password-for-scott> M1
Creating OraclePool
Done creating OraclePool
Getting an Oracle from OraclePool
Done
Is logical connection:true
Getting an Oracle from OraclePool
Done
Is logical connection:true
```

D.4 Example Queries Using Graph Adapter for Eclipse RDF4J

This section describes example queries for using Oracle RDF Graph Adapter for Eclipse RDF4J in an existing MDSYS network.

To run a query, you must do the following:

1. Include any example code described in Example Queries Using Oracle RDF Graph Adapter for Eclipse RDF4J in a Java source file.
2. Define a `CLASSPATH` environment variable named `CP` to include the relevant jar files. For example, it may be defined as follows:

   ```
   setenv CP .:ojdbc8.jar:ucp.jar:oracle-rdf4j-adapter-4.2.1.jar:log4j-
   api-2.17.2.jar:log4j-core-2.17.2.jar:log4j-slf4j-impl-2.17.2.jar:slf4j-
   api-1.7.36.jar:eclipse-rdf4j-4.2.1-onejar.jar:commons-io-2.11.0.jar
   ```

   > **Note:**
   >
   > The preceding `setenv` command assumes that the jar files are located in the current directory. You may need to alter the command to indicate the location of these jar files in your environment.

3. Compile the Java source file. For example, to compile the source file `Test.java`, run the following command:

   ```
   javac -classpath $CP Test.java
   ```

4. Run the compiled file on an RDF graph (model) named `TestModel` in an existing MDSYS network by executing the following command:

   ```
   java -classpath $CP Test jdbc:oracle:thin:@localhost:1521:orcl scott
   <password-for-scott> TestModel
   ```

D.5 Reference Information (MDSYS_Owned Semantic Network Only)

This section provides reference information about RDF Semantic Graph subprograms that apply only for MDSYS-owned semantic networks.

- SEM_OLS Package Subprograms
  The SEM_OLS package contains subprograms (functions and procedures) related to triple-level security to RDF data, using Oracle Label Security (OLS).
- SEM_APIS.PRIVILEGE_ON_APP_TABLES
- SEM_APIS.REMOVE_DUPLICATES

D.5.1 SEM_OLS Package Subprograms

The SEM_OLS package contains subprograms (functions and procedures) related to triple-level security to RDF data, using Oracle Label Security (OLS).

To use the subprograms in this chapter, you should understand the conceptual and usage information in RDF Semantic Graph Overview and Fine-Grained Access Control for RDF Data.

This chapter provides reference information about the subprograms, listed in alphabetical order.

- SEM_OLS.APPLY_POLICY_TO_APP_TAB
- SEM_OLS.REMOVE_POLICY_FROM_APP_TAB

D.5.1.1 SEM_OLS.APPLY_POLICY_TO_APP_TAB

Format

```
SEM_OLS.APPLY_POLICY_TO_APP_TAB(
     policy_name    IN VARCHAR2,
     schema_name    IN VARCHAR2,
     table_name     IN VARCHAR2,
     predicate      IN VARCHAR2 DEFAULT NULL);
```

Description

Applies an OLS policy to an application table in the MDSYS-owned network.

Parameters

policy_name
Name of an existing OLS policy.

schema_name
Name of the schema containing the application table.

table_name
Name of the application table.

predicate
An additional predicate to combine with the label-based predicate.

Usage Notes

When you use triple-level security, OLS is applied to each semantic model in the network. That is, label security is applied to the relevant internal tables and to all the application tables; there is no need to manually apply policies to the application tables of existing semantic models. However, if you need to create additional models after applying the OLS policy, you must use the SEM_OLS.APPLY_POLICY_TO_APP_TAB procedure to apply OLS to the application table before creating the model.

You must have the following to execute this procedure: `EXECUTE` privilege for the SA_POLICY_ADMIN package, and the `policy_DBA` role.

Before executing this procedure, you must have executed the SEM_RDFSA.APPLY_OLS_POLICY procedure specifying `SEM_RDFSA.TRIPLE_LEVEL_ONLY` for the `rdfsa_options` parameter.

To remove the OLS policy from the application table, use the SEM_OLS.REMOVE_POLICY_FROM_APP_TAB procedure.

For information about support for OLS, see Fine-Grained Access Control for RDF Data.

This procedure applies only to the MDSYS-owned network, not to schema-private networks. (If you try to apply this procedure to a schema-private network, the error "ORA-20000: No application tables for schema-private network" is returned.) For information about semantic network types and options, see RDF Networks.

Examples

The following example applies an OLS policy named `defense` to the MY_SCHEMA.MY_APP_TABLE application table.

```
begin
  sem_ols.apply_policy_to_app_table(
        policy_name => 'defense',
        schema_name => 'my_schema',
        table_name  => 'my_app_table');
end;
/
```

D.5.1.2 SEM_OLS.REMOVE_POLICY_FROM_APP_TAB

Format

```
SEM_OLS.REMOVE_POLICY_FROM_APP_TAB(
     policy_name    IN VARCHAR2,
     schema_name    IN VARCHAR2,
     table_name     IN VARCHAR2,
     check_model    IN BOOLEAN DEFAULT TRUE);
```

Description

Permanently removes or detaches the OLS policy from an application table associated with a model in the MDSYS-owned network.

Parameters

policy_name
Name of the existing OLS policy.

schema_name
Name of the schema containing the application table.

table_name
Name of the application table.

check_model
TRUE (the default) checks if the model associated with the application table exists (and generates an exception if the model exists); FALSE does not check if the model exists before performing the operation.

Usage Notes

If you have dropped a semantic model and you no longer need to protect the application table, you can use this procedure.

You must have the following to execute this procedure: EXECUTE privilege for the SA_POLICY_ADMIN package, and the policy_DBA role.

Before executing this procedure, you must have executed the SEM_RDFSA.APPLY_OLS_POLICY procedure specifying SEM_RDFSA.TRIPLE_LEVEL_ONLY for the rdfsa_options parameter.

If check_model is TRUE (the default), an exception is generated if the associated model exists. In this case, if you want to execute this procedure, you must first drop the model.

For information about support for OLS, see Fine-Grained Access Control for RDF Data.

This procedure applies only to the MDSYS-owned network, not to schema-private networks. (If you try to apply this procedure to a schema-private network, the error "ORA-20000: No application tables for schema-private network" is returned.) For information about semantic network types and options, see RDF Networks.

Examples

The following example removes the OLS policy named defense from the MY_SCHEMA.MY_APP_TABLE application table.

```
begin
  sem_ols.remove_policy_from_app_table(
        policy_name => 'defense',
        schema_name => 'my_schema',
        table_name  => 'my_app_table');
end;
/
```

D.5.2 SEM_APIS.PRIVILEGE_ON_APP_TABLES

Format

```
SEM_APIS.PRIVILEGE_ON_APP_TABLES(
     command       IN VARCHAR2 DEFAULT 'GRANT',
     privilege     IN VARCHAR2 DEFAULT 'SELECT',
     network_owner IN VARCHAR2 DEFAULT NULL,
     network_name  IN VARCHAR2 DEFAULT NULL);
```

Description

Grants (or revokes) SELECT or INSERT privilege to (or from) MDSYS on application tables corresponding to all the RDF models owned by the invoker.

Parameters

command
SQL statement, with possible values `GRANT` (the default) or `REVOKE` (case insensitive).

privilege
Privilege name, with possible values `SELECT` (the default) or `INSERT` (case insensitive).

network_owner

Owner of the semantic network. (See Table 1-2.)

network_name

Name of the semantic network. (See Table 1-2.)

Usage Notes

For information about semantic network types and options, see RDF Networks.

Examples

The following example grants SELECT privilege to MDSYS on application tables corresponding to all the RDF models owned by the invoker.

```
EXECUTE SEM_APIS.PRIVILEGE_ON_APP_TABLES('grant', 'select');
```

D.5.3 SEM_APIS.REMOVE_DUPLICATES

Format

```
SEM_APIS.REMOVE_DUPLICATES(
     model_name           IN VARCHAR2,
     threshold            IN FLOAT DEFAULT 0.3,
     rebuild_apptab_index IN BOOLEAN DEFAULT TRUE,
     options              IN VARCHAR2 DEFAULT NULL,
     network_owner        IN VARCHAR2 DEFAULT NULL,
     network_name         IN VARCHAR2 DEFAULT NULL);
```

Description

Removes duplicate triples from a model.

Parameters

model_name
Name of the model.

threshold
A value to determine how numerous triples must be in order for the removal operation to be performed. This procedure removes triples only if the number of triples in the model exceeds the following formula: (total-triples - total-unique-triples + 0.01) / (total-unique-triples + 0.01). For the default value of 0.3 and a model containing 1000 total triples (including duplicates), duplicate triples would be removed only if the number of duplicates exceeds approximately 230.

The lower the threshold value, the fewer duplicates are needed for the procedure to remove duplicates; the higher the threshold value, the more duplicates are needed for the procedure to remove duplicates.

rebuild_apptab_index
`TRUE` (the default) causes all usable indexes on tables that were affected by this operation to be rebuilt after the duplicate triples are removed; `FALSE` does not rebuild any indexes.

options
(Reserved for future use.)

network_owner
Owner of the semantic network. (See Table 1-2.)

network_name
Name of the semantic network. (See Table 1-2.)

Usage Notes

When duplicate triples are removed, all information in the removed rows is lost, including information in columns other than the triple column.

This procedure is not supported on virtual models (explained in Virtual Models).

If the model is empty, or if it contains no duplicate triples or not enough duplicate triples (as computed using the `threshold` value), this procedure does not perform any removal operations.

If there are not enough duplicates (as computed using the `threshold` value) to perform the operation, an informational message is displayed.

If unusable indexes are involved, be sure that the SKIP_UNUSABLE_INDEXES system parameter is set to `TRUE`. Although `TRUE` is the default value for this parameter, some production databases may use the value `FALSE`; therefore, if you need to change it, enter the following:

```
SQL> alter session set skip_unusable_indexes=true;
```

To use this procedure on an application table with one or more user-defined triggers, you must connect as a DBA user and grant the ALTER ANY TRIGGER privilege to the MDSYS user, as follows:

```
SQL> grant alter any trigger to MDSYS;
```

For information about semantic network types and options, see RDF Networks.

Examples

The following example removes duplicate triples in the model named `family`. It accepts the default threshold value of 0.3 and (by default) rebuilds indexes after the duplicates are removed.

```
EXECUTE SEM_APIS.REMOVE_DUPLICATES('family');
```

D.6 Migrating an MDSYS-Owned Network to a Schema-Private Network

You can migrate an MDSYS-owned semantic network in a database to a schema-private semantic network in the same database.

The following example migrates an existing MDSYS semantic network to a shared schema-private semantic network by using SEM_APIS.MOVE_SEM_NETWORK_DATA and SEM_APIS.APPEND_SEM_NETWORK_DATA.

Example D-17 Migrating an MDSYS Semantic Network to a Shared Schema-Private Semantic Network

This example performs the following major actions.

1. Creates a database user (RDFEXPIMPU), if it does not already exist in the database, that will hold the moved existing MDSYS-owned semantic network.
2. Moves the existing semantic network data to the RDFEXPIMPU schema.
3. Creates a administrative database user (RDFADMIN), if it does not already exist in the database, that will own the schema-private semantic network.
4. Creates the schema-private semantic network, named MY_NET and owned by RDFADMIN.
5. Sets up network sharing for this newly created schema-private network.
   a. Grants network sharing privileges to RDFADMIN.
   b. Enables network sharing for all users of the old MDSYS-owned network.
   c. Grants access privileges to two regular database users (UDFUSER and DB_USER1). privileges to RDFADMIN.
6. Appends the previously moved network data into the shared schema-private semantic network.

```
conn sys/<password_for_sys>

-- create a user to hold the moved semantic network
grant connect, resource, unlimited tablespace to rdfexpimpu identified by
rdfexpimpu;

conn system/<password_for_system>

-- move the existing MDSYS semantic network
exec sem_apis.move_sem_network_data(dest_schema=>'RDFEXPIMPU');

-- drop the existing MDSYS semantic network
exec sem_apis.drop_sem_network(cascade=>true);

-- create schema-private semantic network to hold the MDSYS network data
conn sys/<password_for_sys>

-- create an admin user to own the schema-private semantic network
create user rdfadmin identified by rdfadmin;
grant connect,resource,unlimited tablespace to rdfadmin;
```

```
conn system/<password_for_system>

-- create the schema-private semantic network
exec
sem_apis.create_sem_network(tablespace_name=>'rdf_tablespace',network_owner=>'
RDFADMIN',network_name=>'MYNET');

-- setup network sharing for rdfadmin’s schema-private semantic network
-- first grant network sharing privileges to rdfadmin
exec sem_apis.grant_network_sharing_privs(network_owner=>'RDFADMIN');
-- now connect as rdfadmin and enable sharing for all users of the old MDSYS
semantic network
conn rdfadmin/<password>
-- enable sharing for rdfadmin’s network
exec
sem_apis.enable_network_sharing(network_owner=>'RDFADMIN',network_name=>'MYNET
');

-- grant access privileges to RDFUSER
exec
sem_apis.grant_network_access_privs(network_owner=>'RDFADMIN',network_name=>'M
YNET',network_user=>'RDFUSER');
-- grant access privileges to DB_USER1
exec
sem_apis.grant_network_access_privs(network_owner=>'RDFADMIN',network_name=>'M
YNET',network_user=>'DB_USER1');

-- append the exported network into the shared schema-private semantic network
-- after this step, migration will be complete, and the new shared schema-
private semantic network will be ready to use
conn system/<password_for_system>
exec
sem_apis.append_sem_network_data(from_schema=>'RDFEXPIMPU',network_owner=>'RDF
ADMIN',network_name=>'MYNET');
```

E

Changes in Terminology and Subprograms

This appendix introduces changes to a few RDF terminologies and reference procedure names (in the SEM_APIS package) that apply from Oracle Database Release 23ai onwards.

The following table describes the RDF terms that have changed in this book.

Table E-1 Changes in Terminology

New Term	Old Term
RDF graph	Semantic model
RDF network	Semantic network
Inferred graph	Entailment
RDF graph collection	Virtual model
Subject-Property-Matrix (SPM) tables	Result tables
Single-Valued Property (SVP) table	Star-Pattern table
Multi-Valued Property (MVP) table	Triple-Pattern table
Property Chain (PCN) table	Chain-Pattern table

In alignment with the new terms described in Table E-1, the names of a few existing reference procedures in the SEM_APIS package have changed as described in the following table. Oracle Graph recommends that you use the new subprograms starting from Oracle Database Release 23ai.

Table E-2 Changes to the Subprogram Names in the SEM_APIS Package

New Subprogram	Old Subprogram
SEM_APIS.ADD_NETWORK_INDEX	SEM_APIS.ADD_SEM_INDEX
SEM_APIS.ALTER_INDEX_ON_INFERRED_GRAPH	SEM_APIS.ALTER_SEM_INDEX_ON_ENTAILMENT
SEM_APIS.ALTER_INDEX_ON_RDF_GRAPH	SEM_APIS.ALTER_SEM_INDEX_ON_MODEL
SEM_APIS.ALTER_INFERRED_GRAPH	SEM_APIS.ALTER_ENTAILMENT
SEM_APIS.ALTER_RDF_GRAPH	SEM_APIS.ALTER_MODEL
SEM_APIS.ALTER_RDF_INDEXES	SEM_APIS.ALTER_SEM_INDEXES
SEM_APIS.ALTER_RESULT_TAB	SEM_APIS.ALTER_SPM_TAB
SEM_APIS.ANALYZE_INFERRED_GRAPH	SEM_APIS.ANALYZE_ENTAILMENT
SEM_APIS.ANALYZE_RDF_GRAPH	SEM_APIS.ANALYZE_MODEL
SEM_APIS.APPEND_RDF_NETWORK_DATA	SEM_APIS.APPEND_SEM_NETWORK_DATA
SEM_APIS.BUILD_RESULT_TAB	SEM_APIS.BUILD_SPM_TAB
SEM_APIS.BULK_LOAD_RDF_GRAPH	SEM_APIS.BULK_LOAD_FROM_STAGING_TABLE
SEM_APIS.CREATE_INDEX_ON_RESULT_TAB	SEM_APIS.CREATE_INDEX_ON_SPM_TAB
SEM_APIS.CREATE_INFERRED_GRAPH	SEM_APIS.CREATE_ENTAILMENT
SEM_APIS.CREATE_RDF_GRAPH	SEM_APIS.CREATE_SEM_MODEL

Table E-2 (Cont.) Changes to the Subprogram Names in the SEM_APIS Package

New Subprogram	Old Subprogram
SEM_APIS.CREATE_RDF_GRAPH_COLLECTION	SEM_APIS.CREATE_VIRTUAL_MODEL
SEM_APIS.CREATE_RDF_NETWORK	SEM_APIS.CREATE_SEM_NETWORK
SEM_APIS.CREATE_RDFVIEW_GRAPH	SEM_APIS.CREATE_RDFVIEW_MODEL
SEM_APIS.DISABLE_INMEMORY_FOR_INF_GRAPH	SEM_APIS.DISABLE_INMEMORY_FOR_ENT
SEM_APIS.DISABLE_INMEMORY_FOR_RDF_GRAPH	SEM_APIS.DISABLE_INMEMORY_FOR_MODEL
SEM_APIS.DROP_NETWORK_INDEX	SEM_APIS.DROP_SEM_INDEX
SEM_APIS.DROP_INFERRED_GRAPH	SEM_APIS.DROP_ENTAILMENT
SEM_APIS.DROP_RDF_GRAPH	SEM_APIS.DROP_SEM_MODEL
SEM_APIS.DROP_RDF_GRAPH_COLLECTION	SEM_APIS.DROP_VIRTUAL_MODEL
SEM_APIS.DROP_RDF_NETWORK	SEM_APIS.DROP_SEM_NETWORK
SEM_APIS.DROP_RDFVIEW_GRAPH	SEM_APIS.DROP_RDFVIEW_MODEL
SEM_APIS.DROP_RESULT_TAB	SEM_APIS.DROP_SPM_TAB
SEM_APIS.ENABLE_INMEMORY_FOR_INF_GRAPH	SEM_APIS.ENABLE_INMEMORY_FOR_ENT
SEM_APIS.ENABLE_INMEMORY_FOR_RDF_GRAPH	SEM_APIS.ENABLE_INMEMORY_FOR_MODEL
SEM_APIS.EXPORT_RDFVIEW_GRAPH	SEM_APIS.EXPORT_RDFVIEW_MODEL
SEM_APIS.GRANT_RDF_GRAPH_ACCESS_PRIV	SEM_APIS.GRANT_MODEL_ACCESS_PRIV
SEM_APIS.GRANT_RDF_GRAPH_ACCESS_PRIVS	SEM_APIS.GRANT_MODEL_ACCESS_PRIVS
SEM_APIS.MERGE_RDF_GRAPHS	SEM_APIS.MERGE_MODELS
SEM_APIS.MOVE_RDF_NETWORK_DATA	SEM_APIS.MOVE_SEM_NETWORK_DATA
SEM_APIS.REFRESH_NETWORK_INDEX_INFO	SEM_APIS.REFRESH_SEM_NETWORK_INDEX_INFO
SEM_APIS.RENAME_INFERRED_GRAPH	SEM_APIS.RENAME_ENTAILMENT
SEM_APIS.RENAME_RDF_GRAPH	SEM_APIS.RENAME_MODEL
SEM_APIS.RESTORE_RDF_NETWORK_DATA	SEM_APIS.RESTORE_SEM_NETWORK_DATA
SEM_APIS.REVOKE_RDF_GRAPH_ACCESS_PRIV	SEM_APIS.REVOKE_MODEL_ACCESS_PRIV
SEM_APIS.REVOKE_RDF_GRAPH_ACCESS_PRIVS	SEM_APIS.REVOKE_MODEL_ACCESS_PRIVS
SEM_APIS.TRUNCATE_RDF_GRAPH	SEM_APIS.TRUNCATE_SEM_MODEL
SEM_APIS.UPDATE_RDF_GRAPH	SEM_APIS.UPDATE_MODEL
SEM_APIS.VALIDATE_INFERRED_GRAPH	SEM_APIS.VALIDATE_ENTAILMENT
SEM_APIS.VALIDATE_RDF_GRAPH	SEM_APIS.VALIDATE_MODEL

Glossary

apply pattern

Part of a data access constraint defines additional graph patterns to be applied on the resources that match the match pattern before they can be used to construct the query results. *See also*: match pattern

basic graph pattern (BGP)

A set of triple patterns. From the W3C SPARQL Query Language for RDF Recommendation: "SPARQL graph pattern matching is defined in terms of combining the results from matching basic graph patterns. A sequence of triple patterns interrupted by a filter comprises a single basic graph pattern. Any graph pattern terminates a basic graph pattern."

clique

A graph in which every node of it is connected to, bidirectionally, every other node in the same graph.

Cytoscape

An open source bioinformatics software platform for visualizing molecular interaction networks and integrating these interactions with gene expression profiles and other state data. (See `http://www.cytoscape.org/`.) An RDF viewer (available for download) is provided as a Cytoscape plug-in.

entailment

An object containing precomputed triples that can be inferred from applying a specified set of rulebases to a specified set of models. *See also*: rulebase

extractor policy

A named dictionary entity that determines the characteristics of a semantic index that is created using the policy. Each extractor policy refers, directly or indirectly, to an instance of an extractor type.

graph pattern

A combination of triples constructed by combining triple patterns in various ways, including conjunction of triple patterns into groups, optionally using filter conditions, and then combining such groups using connectors similar to disjunctions, outer-joins, and so on. SPARQL querying is based around graph pattern matching.

inferencing

The ability to make logical deductions based on rules. Inferencing enables you to construct queries that perform semantic matching based on meaningful relationships among pieces of data, as opposed to just syntactic matching based on string or other values. Inferencing involves the use of rules, either supplied by Oracle or user-defined, placed in rulebases.

information extractor

An application that processes unstructured documents and extract meaningful information from them, often using natural-language processing engines with the aid of ontologies.

match pattern

Part of a constraint that determines the type of access restriction it enforces and binds one or more variables to the corresponding data instances accessed in the user query. *See also*: apply pattern

model

A user-created semantic structure that has a model name, and refers to triples stored in a specified table column. Examples in this manual are the Articles and Family models.

ontology

A shared conceptualization of knowledge in a particular domain. It consists of a collection of classes, properties, and optionally instances. Classes are typically related by class hierarchy (subclass/ superclass relationship). Similarly, the properties can be related by property hierarchy (subproperty/ superproperty relationship). Properties can be symmetric or transitive, or both. Properties can also have domain, ranges, and cardinality constraints specified for them.

OWLPrime

An Oracle-defined subset of OWL capabilities; refers to the elements of the OWL standard supported by the RDF Semantic Graph native inferencing engine.

RDF Semantic Graph support for Apache Jena

An Oracle-supplied adapter (available for download) for Apache Jena, which is a Java framework for building Semantic Web applications.

reasoning

See inferencing

rule

An object that can be applied to draw inferences from semantic data.

rulebase

An object that can contain rules. *See also*: rule

rules index

See: entailment

semantic index

An index of type MDSYS.SEMCONTEXT, created on textual documents stored in a column of a table, and used with information extractors to locate and extract meaningful information from unstructured documents. *See also*: information extractor

Simple Knowledge Organization System (SKOS)

A data model that is especially useful for representing thesauri, classification schemes, taxonomies, and other types of controlled vocabulary. SKOS is based on standard semantic web technologies including RDF and OWL, which makes it easy to define the formal semantics for those knowledge organization systems and to share the semantics across applications.

triple pattern

Similar to an RDF triple, but allows use of a variable in place of any of the three components (subject, predicate, or object). Triple patterns are basic elements in graph patterns used in SPARQL queries. A triple pattern used in a query against an RDF graph is said to match if, substitution of RDF terms for the variables present in the triple pattern, creates a triple that is present in the RDF graph. *See also*: graph pattern

Index

Symbols

A

B

C

D

H

I

J

L

M

P

Q

R

S