Oracle® Database

Semantic Technologies Developer's Guide

11*g* Release 1 (11.1)

B28397-05

July 2009

Provides usage and reference information about Oracle Database 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).

ORACLE®

Oracle Database Semantic Technologies Developer's Guide, 11*g* Release 1 (11.1)

B28397-05

Copyright © 2005, 2009, Oracle and/or its affiliates. All rights reserved.

Primary Author: Chuck Murray

Contributors: Eugene Inseok Chong, Souri Das, Matt Perry, Jags Srinivasan, Zhe (Alan) Wu

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this software or related documentation is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS Programs, software, databases, and related documentation and technical data delivered to U.S. Government customers are "commercial computer software" or "commercial technical data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, duplication, disclosure, modification, and adaptation shall be subject to the restrictions and license terms set forth in the applicable Government contract, and, to the extent applicable by the terms of the Government contract, the additional rights set forth in FAR 52.227-19, Commercial Computer Software License (December 2007). Oracle USA, Inc., 500 Oracle Parkway, Redwood City, CA 94065.

This software is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications which may create a risk of personal injury. If you use this software in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure the safe use of this software. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software in dangerous applications.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

This software and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Contents

Preface ix

Audience ix
Documentation Accessibility ix
Related Documents x
Conventions x

What's New in Semantic Technologies? xi

Features Added for Release 11.1.0.7 (November, 2008) xi
Storage Model Enhancements and Migration xii
Support for OWL Inferencing xii
New Bulk Loading Interface for Improved Performance xii
Ontology-Assisted Querying of Relational Data xii
Required Procedure for Using Semantic Technology Support xii

1 Oracle Semantic Technologies Overview

1.1 Introduction to Oracle Semantic Technologies 1-2
1.2 Semantic Data Modeling 1-3
1.3 Semantic Data in the Database 1-3
1.3.1 Metadata for Models 1-3
1.3.2 Statements 1-5
1.3.2.1 Triple Uniqueness and Data Types for Literals 1-6
1.3.3 Subjects and Objects 1-7
1.3.4 Blank Nodes 1-7
1.3.5 Properties 1-7
1.3.6 Inferencing: Rules and Rulebases 1-7
1.3.7 Rules Indexes 1-10
1.3.8 Virtual Models 1-11
1.3.9 Semantic Data Security Considerations 1-13
1.4 Semantic Metadata Tables and Views 1-14
1.5 Semantic Data Types, Constructors, and Methods 1-14
1.5.1 Constructors for Inserting Triples Without Any Blank Nodes 1-16
1.5.2 Constructors for Inserting Triples With or Without Any Blank Nodes 1-16
1.6 Using the SEM_MATCH Table Function to Query Semantic Data 1-17
1.6.1 Performing Queries with Incomplete or Invalid Rules Indexes 1-20
1.6.2 Graph Patterns: Support for Curly Brace Syntax and OPTIONAL Keyword 1-21

1.7 Loading and Exporting Semantic Data 1-22
1.7.1 Bulk Loading Semantic Data Using a Staging Table 1-22
1.7.1.1 Recording Event Traces during Bulk Loading 1-23
1.7.2 Batch Loading Semantic Data Using the Java API 1-23
1.7.2.1 When to Choose Batch Loading 1-24
1.7.3 Loading Semantic Data Using INSERT Statements 1-25
1.7.4 Exporting Semantic Data 1-25
1.8 Creating and Managing Semantic Network Indexes 1-25
1.9 Quick Start for Using Semantic Data 1-26
1.10 Semantic Data Examples 1-27
1.10.1 Example: Journal Article Information 1-28
1.10.2 Example: Family Information 1-29
1.11 Required Procedure for Semantic Technologies Support 1-36
1.12 Partitioning Requirement 1-37
1.13 Downgrading to the Previous Oracle Database Release 1-37
1.14 Software Naming Changes for Semantic Technologies 1-38

2 OWL Concepts

2.1 Ontologies 2-1
2.1.1 Example: Cancer Ontology 2-1
2.1.2 Supported OWL Subsets 2-2
2.2 Using OWL Inferencing 2-4
2.2.1 Creating a Simple OWL Ontology 2-4
2.2.2 Performing Native OWL inferencing 2-5
2.2.3 Performing OWL and User-Defined Rules inferencing 2-5
2.2.4 Generating OWL inferencing Proofs 2-6
2.2.5 Validating OWL Models and Entailments 2-7
2.2.6 Using SEM_APIS.CREATE_ENTAILMENT for RDFS Inference 2-8
2.2.7 Enhancing Inference Performance 2-8
2.2.8 Performing Selective Inferencing (Advanced Information) 2-8
2.3 Using Semantic Operators to Query Relational Data 2-9
2.3.1 Using the SEM_RELATED Operator 2-10
2.3.2 Using the SEM_DISTANCE Ancillary Operator 2-11
2.3.2.1 Computation of Distance Information 2-12
2.3.3 Creating a Semantic Index of Type MDSYS.SEM_INDEXTYPE 2-13
2.3.4 Using SEM_RELATED and SEM_DISTANCE When the Indexed Column Is Not the First Parameter 2-13
2.3.5 Using URIPREFIX When Values Are Not Stored as URIs 2-14

3 SEM_APIS Package Subprograms

SEM_APIS.ADD_SEM_INDEX 3-2
SEM_APIS.ALTER_SEM_INDEX_ON_MODEL 3-3
SEM_APIS.ALTER_SEM_INDEX_ON_RULES_INDEX 3-4
SEM_APIS.ANALYZE_MODEL 3-5
SEM_APIS.ANALYZE_RULES_INDEX 3-7
SEM_APIS.BULK_LOAD_FROM_STAGING_TABLE 3-9

SEM_APIS.CLEANUP_FAILED 3-11
SEM_APIS.CREATE_ENTAILMENT 3-12
SEM_APIS.CREATE_RULEBASE 3-15
SEM_APIS.CREATE_RULES_INDEX 3-16
SEM_APIS.CREATE_SEM_MODEL 3-17
SEM_APIS.CREATE_SEM_NETWORK 3-18
SEM_APIS.CREATE_VIRTUAL_MODEL 3-19
SEM_APIS.DROP_ENTAILMENT 3-21
SEM_APIS.DROP_RULEBASE 3-22
SEM_APIS.DROP_RULES_INDEX 3-23
SEM_APIS.DROP_SEM_INDEX 3-24
SEM_APIS.DROP_SEM_MODEL 3-25
SEM_APIS.DROP_SEM_NETWORK 3-26
SEM_APIS.DROP_USER_INFERENCE_OBJS 3-27
SEM_APIS.DROP_VIRTUAL_MODEL 3-28
SEM_APIS.GET_MODEL_ID 3-29
SEM_APIS.GET_MODEL_NAME 3-30
SEM_APIS.GET_TRIPLE_ID 3-31
SEM_APIS.IS_TRIPLE 3-33
SEM_APIS.LOOKUP_RULES_INDEX 3-35
SEM_APIS.VALIDATE_ENTAILMENT 3-36
SEM_APIS.VALIDATE_MODEL 3-38
SEM_APIS.VALUE_NAME_PREFIX 3-40
SEM_APIS.VALUE_NAME_SUFFIX 3-42

4 SEM_PERF Package Subprograms

SEM_PERF.GATHER_STATS 4-2

Index

List of Examples

1–1 Inserting a Rule into a Rulebase 1-9
1–2 Using Rulebases for Inferencing 1-10
1–3 Creating a Rules Index 1-11
1–4 Querying a Virtual Model 1-12
1–5 SDO_RDF_TRIPLE_S Methods 1-15
1–6 SDO_RDF_TRIPLE_S Constructor to Insert a Triple 1-16
1–7 SDO_RDF_TRIPLE_S Constructor to Reusing a Blank Node 1-17
1–8 SEM_MATCH Table Function 1-19
1–9 HINT0 Option with SEM_MATCH Table Function 1-20
1–10 SEM_MATCH Table Function 1-20
1–11 Curly Brace Syntax 1-21
1–12 Curly Brace Syntax and OPTIONAL Construct 1-21
1–13 Using a Model for Journal Article Information 1-28
1–14 Using a Model for Family Information 1-30
2–1 Creating a Simple OWL Ontology 2-4
2–2 Performing Native OWL Inferencing 2-5
2–3 Performing OWL and User-Defined Rules Inferencing 2-5
2–4 Displaying Proof Information 2-6
2–5 Validating an Entailment 2-7
2–6 Performing Selective Inferencing 2-9
2–7 SEM_RELATED Operator 2-10
2–8 SEM_DISTANCE Ancillary Operator 2-11
2–9 Using SEM_DISTANCE to Restrict the Number of Rows Returned 2-12
2–10 Creating a Semantic Index 2-13
2–11 Creating a Semantic Index Specifying a Model and Rulebase 2-13
2–12 Query Benefitting from Generation of Statistical Information 2-13
2–13 Specifying a URI Prefix During Semantic Index Creation 2-14

List of Figures

1–1 Oracle Semantic Capabilities 1-2
1–2 Inferencing 1-8
1–3 Family Tree for RDF Example 1-30
2–1 Cancer Ontology Example 2-2

List of Tables

1–1 MDSYS.SEM_MODEL$ View Columns 1-4
1–2 MDSYS.SEMM_model-name View Columns 1-4
1–3 MDSYS.RDF_VALUE$ Table Columns 1-5
1–4 MDSYS.SEMR_rulebase-name View Columns 1-9
1–5 MDSYS.SEM_RULEBASE_INFO View Columns 1-9
1–6 MDSYS.SEM_RULES_INDEX_INFO View Columns 1-10
1–7 MDSYS.SEM_RULES_INDEX_DATASETS View Columns 1-11
1–8 MDSYS.SEM_MODEL$ View Column Explanations for Virtual Models 1-12
1–9 MDSYS.SEM_VMODEL_INFO View Columns 1-13
1–10 MDSYS.SEM_VMODEL_DATASETS View Columns 1-13
1–11 Semantic Metadata Tables and Views 1-14
1–12 Semantic Technology Software Objects: Old and New Names 1-38
2–1 PATIENTS Table Example Data 2-2
2–2 RDFS/OWL Vocabulary Constructs Included in Each Supported Rulebase 2-3
3–1 Inferencing Keywords for inf_components_in Parameter 3-13

Preface

Oracle Database Semantic Technologies Developer's Guide provides usage and reference information about Oracle Database 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).

Note: Oracle Spatial must be installed before you can use any of the RDF and OWL capabilities. Partitioning must also be enabled, as explained in Section 1.12.

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

Our goal is to make Oracle products, services, and supporting documentation accessible to all users, including users that are disabled. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at `http://www.oracle.com/accessibility/`.

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Deaf/Hard of Hearing Access to Oracle Support Services

To reach Oracle Support Services, use a telecommunications relay service (TRS) to call Oracle Support at 1.800.223.1711. An Oracle Support Services engineer will handle technical issues and provide customer support according to the Oracle service request process. Information about TRS is available at `http://www.fcc.gov/cgb/consumerfacts/trs.html`, and a list of phone numbers is available at `http://www.fcc.gov/cgb/dro/trsphonebk.html`.

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.

What's New in Semantic Technologies?

For Oracle Database release 11.1, the focus of this manual expands to include Oracle Database semantic technologies, which include Resource Description Framework (RDF), which was supported in Release 10.2, and a subset of the Web Ontology Language (OWL), support for which is new in the current release.

In addition to expanded capabilities, some naming conventions associated with the API may changes. For example, names that include "SDO_RDF" may be changed for data types, function names, and PL/SQL package names. However, all applications created in the previous release using supported names for that release will continue to run.

> **Note:** These changes are only partially reflected in the content of this draft; however, the organization and the content of this draft will be periodically updated to reflect the new capabilities and names

Features Added for Release 11.1.0.7 (November, 2008)

This section describes features that are included in a Release 11.1.0.7.0 patch that was made available on Oracle *MetaLink* in November, 2008.

- Support for virtual models (see Section 1.3.8)
- Curly brace syntax for SEM_MATCH graph pattern, including support for the OPTIONAL construct (see Section 1.6.2)
- Using HINT0 ("hint-zero") in a SEM_MATCH query (see Section 1.6)
- New columns returned from SEM_MATCH: id, _prefix, _suffix (see Section 1.6)
- Ability to create and manage semantic network indexes on models and rules indexes (see Section 1.8)
- SEM_APIS.BULK_LOAD_FROM_STAGING_TABLE procedure: New options in the `flags` parameter `join_hint` for tasks `IZC`, `MBV`, and `MBT`; `parallel=`*`n`*)
- Simplification of staging table definition (fewer columns) and privilege requirements (see Section 1.7.1)
- Simpler event tracing during bulk load (see the information about the new RDF$ET_TAB table in Section 1.7.1.1)

Storage Model Enhancements and Migration

The storage model has been enhanced to support OWL inferencing: some internal data structures and indexes have been changed, added, and removed. These changes also result in enhanced performance.

Because of the extent of these changes, if you have semantic data that you used with the previous release, you must upgrade that data to migrate it to the new format before you can use any new features for this release. Semantic data is upgraded as part of the required procedure described in Section 1.11.

Support for OWL Inferencing

Support has been added to support storing, validating, and querying Web Ontology Language (OWL)-based ontologies. Support is provided for a subset of the OWL DL language.

To query ontology data, you can use table functions and operators that examine semantic relationships, such as SEM_MATCH, SEM_RELATED, and SEM_DISTANCE.

New Bulk Loading Interface for Improved Performance

You can improve performance for bulk loading of semantic data in bulk using a staging table and calling the SEM_APIS.BULK_LOAD_FROM_STAGING_TABLE procedure. For more information, see Section 1.7.1.

Ontology-Assisted Querying of Relational Data

You can go beyond syntactic matching to perform semantic relatedness-based querying of relational data, by associating an ontology with the data and using the new SEM_RELATED operator (and optionally its SEM_DISTANCE ancillary operator). The new SEM_INDEXTYPE index type improves performance for semantic queries.

Required Procedure for Using Semantic Technology Support

Before you can use any types, synonyms, or PL/SQL packages related to Oracle semantic technologies support, you must run the `$ORACLE_HOME/md/admin/catsem10i.sql` or `$ORACLE_HOME/md/admin/catsem11i.sql` script, as explained in Section 1.11. This procedure installs Release 11.1 support for semantic technologies support, and it migrates any existing Release 10.2 RDF data to the Release 11.1 format.

1

Oracle Semantic Technologies Overview

This chapter describes the support for semantic technologies, specifically Resource Description Framework (RDF) and a subset of the Web Ontology Language (OWL). It assumes that you are familiar with the major concepts associated with RDF and OWL, such as {subject, predicate, object} triples, URIs, blank nodes, plain and typed literals, and ontologies. This chapter 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 semantic data are in the SEM_APIS package, which is documented in Chapter 3.

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

This chapter contains the following major sections:

- Section 1.1, "Introduction to Oracle Semantic Technologies"
- Section 1.2, "Semantic Data Modeling"
- Section 1.3, "Semantic Data in the Database"
- Section 1.4, "Semantic Metadata Tables and Views"
- Section 1.5, "Semantic Data Types, Constructors, and Methods"
- Section 1.6, "Using the SEM_MATCH Table Function to Query Semantic Data"
- Section 1.7, "Loading and Exporting Semantic Data"
- Section 1.9, "Quick Start for Using Semantic Data"
- Section 1.10, "Semantic Data Examples"
- Section 1.11, "Required Procedure for Semantic Technologies Support"
- Section 1.12, "Partitioning Requirement"
- Section 1.13, "Downgrading to the Previous Oracle Database Release"
- Section 1.14, "Software Naming Changes for Semantic Technologies"

For information about OWL concepts and the Oracle Database support for OWL capabilities, see Chapter 2.

Required Script: Before performing any operations described in this guide, you must run the `$ORACLE_HOME/md/admin/catsem10i.sql` or `$ORACLE_HOME/md/admin/catsem11i.sql` script, as explained in Section 1.11.

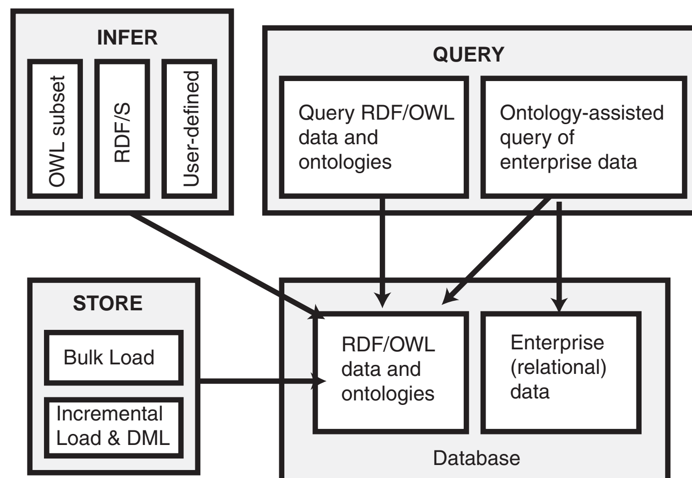
Name Changes: Effective with Oracle Database Release 11.1, the names of many software objects (PL/SQL packages, functions and procedures, system tables and views, and so on) have been changed. In most cases, the change involves replacing the string *RDF* with *SEM*. However, existing applications using valid names from a previous release will continue to work. For more information, see Section 1.14.

Spatial and Partitioning: Oracle Spatial must be installed before you can use any of the RDF and OWL capabilities. Partitioning must also be enabled, as explained in Section 1.12.

1.1 Introduction to Oracle Semantic Technologies

Oracle Database enables you to store semantic data and ontologies, to query semantic 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 semantic data. Figure 1–1 shows how these capabilities interact.

Figure 1–1 Oracle Semantic Capabilities



As shown in Figure 1–1, the database contains semantic data and ontologies (RDF/OWL models), as well as traditional relational data. To load semantic 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 semantic data from a release before Oracle Database 11.1, the data must be upgraded as described in Section 1.11.

You can query semantic data and ontologies, and you can also perform ontology-assisted queries of semantic and traditional relational data to find semantic relationships. To perform ontology-assisted queries, use the SEM_RELATED operator, which is described in Section 2.3.

You can expand the power of queries on semantic 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 Section 1.3.6.

1.2 Semantic Data Modeling

In addition to its formal semantics, semantic 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 a semantic data network. In addition, information is maintained about specific semantic data models created by database users. A user-created **model** has a model 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.

1.3 Semantic Data in the Database

There is one universe for all semantic data stored in the database. All triples are parsed and stored in the system as entries in tables under the MDSYS 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 semantic 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 semantic 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.)

1.3.1 Metadata for Models

The MDSYS.SEM_MODEL$ view contains information about all models defined in the database. When you create a model using the SEM_APIS.CREATE_SEM_MODEL

procedure, you specify a name for the model, as well as a table and column to hold references to the semantic data, and the system automatically generates a model ID.

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

The MDSYS.SEM_MODEL$ view contains the columns shown in Table 1–1.

Table 1–1 MDSYS.SEM_MODEL$ View Columns

Column Name	Data Type	Description
OWNER	VARCHAR2(30)	Schema of the owner of the model.
MODEL_ID	NUMBER	Unique model ID number, automatically generated.
MODEL_NAME	VARCHAR2(25)	Name of the model.
TABLE_NAME	VARCHAR2(30)	Name of the table to hold references to semantic data for the model.
COLUMN_NAME	VARCHAR2(30)	Name of the column of type SDO_RDF_TRIPLE_S in the table to hold references to semantic data for the model.
MODEL_TABLESPACE_NAME	VARCHAR2(30)	Name of the tablespace to be used for storing the triples for this model.

When you create a model, a view for the triples associated with the model is also created under the MDSYS schema. This view has a name in the format RDFM_*model-name*, and it is visible only to the owner of the model and to users with suitable privileges. Each MDSYS.SEMM_*model-name* view contains a row for each triple (stored as a link in a network), and it has the columns shown in Table 1–2.

Table 1–2 MDSYS.SEMM_model-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. It logically partitions the table by RDF graphs.
COST	NUMBER	(Reserved for future use)
CTXT1	NUMBER	(Reserved for future use)
CTXT2	VARCHAR2(4000)	(Reserved for future use)
DISTANCE	NUMBER	(Reserved for future use)
EXPLAIN	VARCHAR2(4000)	(Reserved for future use)
PATH	VARCHAR2(4000)	(Reserved for future use)

Table 1–2 (Cont.) MDSYS.SEMM_model-name View Columns

Column Name	Data Type	Description
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 1–2, for columns P_VALUE_ID, START_NODE_ID, END_NODE_ID, and CANON_END_NODE_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.

1.3.2 Statements

The MDSYS.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 MDSYS.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 1–3.

Table 1–3 MDSYS.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(4000)	If the length of the lexical value is 4000 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#`.
VNAME_SUFFIX	VARCHAR2(512)	If the length of the lexical value is 4000 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`.

Table 1–3 (Cont.) MDSYS.RDF_VALUE$ Table Columns

Column Name	Data Type	Description
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.)
LONG_VALUE	CLOB	The character string if the length of the lexical value is greater than 4000 bytes. Otherwise, this column has a null value.
VALUE_NAME	VARCHAR2(4000)	This is a computed column. If length of the lexical value is 4000 bytes or less, the value of this column is the concatenation of the values of VNAME_PREFIX column and the VNAME_SUFFIX column.

1.3.2.1 Triple Uniqueness and Data Types for Literals

Duplicate triples are not stored in the database. 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 model. 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.

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 literal in the RDF_VALUE$ table is taken as the canonical form and given the VALUE_TYPE value of `CPL`, `CPL@`, `CTL`, `CPLL`, `CPLL@`, or `CTLL` as appropriate; that is, a `C` for canonical is prefixed to the actual value type. If a literal with the same canonical form (but a different lexical representation) as a previously inserted literal is inserted into the RDF_VALUE$ table, the VALUE_TYPE value assigned to the new insert is `PL`, `PL@`, `TL`, `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.

1.3.3 Subjects and Objects

RDF subjects and objects are mapped to nodes in a semantic data 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.

1.3.4 Blank Nodes

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

In an Oracle semantic 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 a semantic model, and never reused between two different models. Thus, when inserting triples involving blank nodes into a model, you must use the SDO_RDF_TRIPLE_S constructor that supports reuse of blank nodes.

1.3.5 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.

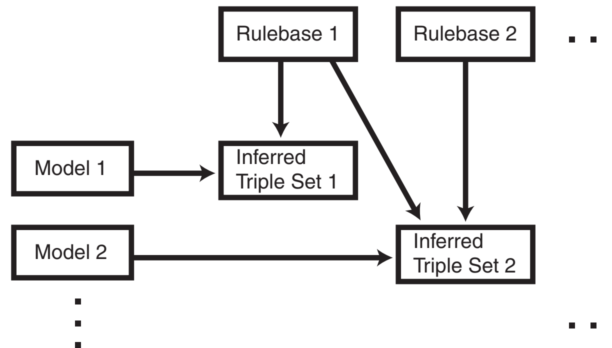
When a triple is inserted into a model, the subject, property, and object text values are checked to see if they already exist in the database. If they already exist (due to previous statements in other models), no new entries are made; if they do not exist, three new rows are inserted into the RDF_VALUE$ table (described in Section 1.3.2).

1.3.6 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 model data and the application of rules in one or more rulebases. In this illustration, the database can have any number of semantic models, 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 semantic data. A rule is identified by a name and consists of:

- An IF side pattern for the antecedents
- An optional filter condition that further restricts the subgraphs matched by the IF side pattern
- 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/'))
)
```

In this case, the rule does not have a filter condition, so that component of the representation is NULL. Note that a THEN side pattern with more than one triple can be used to infer multiple triples for each IF side match.

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)
- OWLPRIME (empty)

The RDFS and RDF rulebases are created when you call the SEM_APIS.CREATE_SEM_NETWORK procedure to add RDF support to the database. The RDFS rulebase implements the RDFS entailment 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 entailment rules, which are a subset of the RDFS

entailment rules. You can see the contents of these rulebases by examining the MDSYS.SEMR_RDFS and MDSYS.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 system table is created to hold rules in the rulebase, along with a system view with a name in the format MDSYS.SEMR_*rulebase-name* (for example, MDSYS.SEMR_FAMILY_RB for a rulebase named `FAMILY_RB`). You must use this view to insert, delete, and modify rules in the rulebase. Each MDSYS.SEMR_*rulebase-name* view has the columns shown in Table 1–4.

Table 1–4 MDSYS.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)	Filter condition that further restricts the subgraphs matched by the IF side pattern. Null indicates no filter condition is to be applied.
CONSEQUENTS	VARCHAR2(4000)	THEN side pattern for the consequents
ALIASES	SEM_ALIASES	One or more namespaces to be used. (The SEM_ALIASES data type is described in Section 1.6.)

Information about all rulebases is maintained in the MDSYS.SEM_RULEBASE_INFO view, which has the columns shown in Table 1–5 and one row for each rulebase.

Table 1–5 MDSYS.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 1–1 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 of (that is, has the `grandParentOf` relationship with respect to) his or her child's child. It also specifies a namespace to be used. (This example is an excerpt from Example 1–14 in Section 1.10.2.)

Example 1–1 Inserting a Rule into a Rulebase

```
EXECUTE SEM_APIS.CREATE_RULEBASE('family_rb');

INSERT INTO mdsys.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/')));
```

You can specify one or more rulebases when calling the SEM_MATCH table function (described in Section 1.6), to control the behavior of queries against semantic data. Example 1–2 refers to the `family_rb` rulebase and to the `grandParentOf` relationship created in Example 1–1, to find all grandfathers (grandparents who are male) and their grandchildren. (This example is an excerpt from Example 1–14 in Section 1.10.2.)

Example 1–2 Using Rulebases for Inferencing

```
-- Select all grandfathers and their grandchildren from the family model.
-- Use inferencing from both the RDFS and family_rb rulebases.
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));
```

For information about support for native OWL inferencing, see Section 2.2.

1.3.7 Rules Indexes

A **rules index** is an object containing precomputed triples that can be inferred from applying a specified set of rulebases to a specified set of models. If a SEM_MATCH query refers to any rulebases, a rules index must exist for each rulebase-model combination in the query.

To create a rules index, use the SEM_APIS.CREATE_RULES_INDEX procedure. To drop (delete) a rules index, use the SEM_APIS.DROP_RULES_INDEX procedure.

When you create a rules index, a view for the triples associated with the rules index is also created under the MDSYS schema. This view has a name in the format SEMI_*rules-index-name*, and it is visible only to the owner of the rules index and to users with suitable privileges. Each MDSYS.SEMI_*rules-index-name* view contains a row for each triple (stored as a link in a network), and it has the same columns as the SEMM_*model-name* view, which is described in Table 1–2 in Section 1.3.1.

Information about all rules indexes is maintained in the MDSYS.SEM_RULES_INDEX_INFO view, which has the columns shown in Table 1–6 and one row for each rules index.

Table 1–6 MDSYS.SEM_RULES_INDEX_INFO View Columns

Column Name	Data Type	Description
OWNER	VARCHAR2(30)	Owner of the rules index
INDEX_NAME	VARCHAR2(25)	Name of the rules index
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 rules index

Table 1–6 (Cont.) MDSYS.SEM_RULES_INDEX_INFO View Columns

Column Name	Data Type	Description
STATUS	VARCHAR2(30)	Contains `VALID` if the rules index is valid, `INVALID` if the rules index is not valid, `INCOMPLETE` if the rules index is incomplete (similar to `INVALID` but requiring less time to re-create), `INPROGRESS` if the rules index is being created, or `FAILED` if a system failure occurred during the creation of the rules index.
MODEL_COUNT	NUMBER	Number of models included in the rules index
RULEBASE_ COUNT	NUMBER	Number of rulebases included in the rules index

Information about all database objects, such as models and rulebases, related to rules indexes is maintained in the MDSYS.SEM_RULES_INDEX_DATASETS view. This view has the columns shown in Table 1–7 and one row for each unique combination of values of all the columns.

Table 1–7 MDSYS.SEM_RULES_INDEX_DATASETS View Columns

Column Name	Data Type	Description
INDEX_NAME	VARCHAR2(25)	Name of the rules index
DATA_TYPE	VARCHAR2(8)	Type of data included in the rules index. Examples: `MODEL` and `RULEBASE`
DATA_NAME	VARCHAR2(25)	Name of the object of the type in the DATA_TYPE column

Example 1–3 creates a rules index named `family_rb_rix_family`, using the `family` model and the `RDFS` and `family_rb` rulebases. (This example is an excerpt from Example 1–14 in Section 1.10.2.)

Example 1–3 Creating a Rules Index

```
BEGIN
  SEM_APIS.CREATE_RULES_INDEX(
    'rdfs_rix_family',
    SEM_Models('family'),
    SEM_Rulebases('RDFS','family_rb'));
END;
/
```

1.3.8 Virtual Models

A virtual model is a logical graph that can be used in a SEM_MATCH query. A virtual model is the result of a UNION or UNION ALL operation on one or more models and optionally the corresponding rules index.

Using a virtual model can simplify management of access privileges for semantic data. For example, assume that you have created three semantic models and one rules index based on the three models and the OWLPRIME rulebase. Without a virtual model, you must individually grant and revoke access privileges for each model and the rules index. However, if you create a virtual model that contains the three models and the rules index, you will only need to grant and revoke access privileges for the single virtual model.

Using a virtual model can also facilitate rapid updates to semantic models. For example, assume that virtual model VM1 contains model M1 and rules index R1 (that is, VM1 = M1 UNION ALL R1), and assume that semantic model M1_UPD is a copy of M1 that has been updated with additional triples and that R1_UPD is a rules index created for M1_UPD. Now, to have user queries over VM1 go to the updated model and rules index, you can redefine virtual model VM1 (that is, VM1 = M1_UPD UNION ALL R1_UPD).

To create a virtual model, use the SEM_APIS.CREATE_VIRTUAL_MODEL procedure. To drop (delete) a virtual model, use the SEM_APIS.DROP_VIRTUAL_MODEL procedure. A virtual model is dropped automatically if any of its component models, rulebases, or rules index are dropped.

To query a virtual model, specify the virtual model name in the `models` parameter of the SEM_MATCH table function, as shown in Example 1–4.

Example 1–4 Querying a Virtual Model

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

For information about the SEM_MATCH table function, see Section 1.6, which includes information using certain attributes when querying a virtual model.

When you create a virtual model, an entry is created for it in the MDSYS.SEM_MODEL$ view, which is described in Table 1–1 in Section 1.3.1. However, the values in several of the columns are different for virtual models as opposed to semantic models, as explained in Table 1–8.

Table 1–8 MDSYS.SEM_MODEL$ View Column Explanations for Virtual Models

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

Information about all virtual models is maintained in the MDSYS.SEM_VMODEL_INFO view, which has the columns shown in Table 1–6 and one row for each virtual model.

Table 1–9 MDSYS.SEM_VMODEL_INFO View Columns

Column Name	Data Type	Description
OWNER	VARCHAR2(30)	Owner of the virtual model
VIRTUAL_MODEL_NAME	VARCHAR2(25)	Name of the virtual model
UNIQUE_VIEW_NAME	VARCHAR2(30)	Name of the view that contains unique triples in the virtual model, 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 virtual model
STATUS	VARCHAR2(30)	Contains `VALID` if the associated rules index is valid, `INVALID` if the rules index is not valid, `INCOMPLETE` if the rules index is incomplete (similar to `INVALID` but requiring less time to re-create), `INPROGRESS` if the rules index is being created, `FAILED` if a system failure occurred during the creation of the rules index, or `NORIDX` if no rules index is associated with the virtual model.
MODEL_COUNT	NUMBER	Number of models in the virtual model
RULEBASE_COUNT	NUMBER	Number of rulebases used for the virtual model
RULES_INDEX_COUNT	NUMBER	Number of rules indexes in the virtual model

Information about all objects (models, rulebases, and rules index) related to virtual models is maintained in the MDSYS.SEM_VMODEL_DATASETS view. This view has the columns shown in Table 1–7 and one row for each unique combination of values of all the columns.

Table 1–10 MDSYS.SEM_VMODEL_DATASETS View Columns

Column Name	Data Type	Description
VIRTUAL_MODEL_NAME	VARCHAR2(25)	Name of the virtual model
DATA_TYPE	VARCHAR2(8)	Type of object included in the virtual model. Examples: `MODEL` for a semantic model, `RULEBASE` for a rulebase, or `RULEIDX` for a rules index
DATA_NAME	VARCHAR2(25)	Name of the object of the type in the DATA_TYPE column

1.3.9 Semantic Data Security Considerations

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

- When a model or rules index 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 model or rules index.
- 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 a rules index 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 a model, a user must have DML privileges for the corresponding base table.
- The creator of the base table corresponding to a model 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 a model can grant SELECT privileges on the corresponding database view to other users.
- A user can query only those models for which that user has SELECT privileges to the corresponding database views.
- Only the creator of a model or a rulebase can drop it.

1.4 Semantic Metadata Tables and Views

Oracle Database maintains several tables and views in the MDSYS schema to hold metadata related to semantic data. (Some of these tables and views are created by the SEM_APIS.CREATE_SEM_NETWORK procedure, as explained in Section 1.9, and some are created only as needed.) Table 1–11 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 users with DBA privileges.)

Table 1–11 Semantic Metadata Tables and Views

Name	Contains Information About	Described In
RDF_VALUE$	Subjects, properties, and objects used to represent statements	Section 1.3.2
SEM_MODEL$	All models defined in the database	Section 1.3.1
SEMM_*model-name*	Triples in the specified model	Section 1.3.1
SEM_RULEBASE_INFO	Rulebases	Section 1.3.6
SEM_RULES_INDEX_DATASETS	Database objects used in rules indexes	Section 1.3.7
SEM_RULES_INDEX_INFO	Rules indexes	Section 1.3.7
SEM_VMODEL_INFO	Virtual models	Section 1.3.8
SEM_VMODEL_DATASETS	Database objects used in virtual models	Section 1.3.8
SEMI_*rules-index-name*	Triples in the specified rules index	Section 1.3.7
SEMR_*rulebase-name*	Rules in the specified rulebase	Section 1.3.6
SEMU_*virtual-model-name*	Unique triples in the virtual model	Section 1.3.8
SEMV_*virtual-model-name*	Triples in the virtual model	Section 1.3.8

1.5 Semantic Data Types, Constructors, and Methods

The SDO_RDF_TRIPLE object type represents semantic data in triple format, and the SDO_RDF_TRIPLE_S object type (the _S for storage) stores persistent semantic data in the database. The SDO_RDF_TRIPLE_S type has references to the data, because the

actual semantic 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 model and cannot be reused across models

---

The SDO_RDF_TRIPLE type is used to display triples, whereas the SDO_RDF_TRIPLE_S type is used to store the triples in database tables.

The SDO_RDF_TRIPLE object type has the following attributes:

```
SDO_RDF_TRIPLE (
  subject VARCHAR2(4000),
  property VARCHAR2(4000),
  object VARCHAR2(10000))
```

The SDO_RDF_TRIPLE_S object type has the following attributes:

```
SDO_RDF_TRIPLE_S (
  RDF_C_ID  NUMBER,  -- Canonical object value ID
  SEM_M_ID NUMBER,  -- Model 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 a triple or a part (subject, property, or object) of a triple:

```
GET_TRIPLE() RETURNS SDO_RDF_TRIPLE
GET_SUBJECT() RETURNS VARCHAR2
GET_PROPERTY() RETURNS VARCHAR2
GET_OBJECT() RETURNS CLOB
```

Example 1–5 shows the SDO_RDF_TRIPLE_S methods.

Example 1–5 SDO_RDF_TRIPLE_S Methods

```
SELECT a.triple.GET_TRIPLE() AS triple
  FROM articles_rdf_data a WHERE a.id = 1;

TRIPLE(SUBJECT, PROPERTY, OBJECT)
--------------------------------------------------------------------------------
SDO_RDF_TRIPLE('<http://nature.example.com/Article1>', '<http://purl.org/dc/elem
ents/1.1/title>', '<All about XYZ>')

SELECT a.triple.GET_SUBJECT() AS subject
  FROM articles_rdf_data a WHERE a.id = 1;

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

SELECT a.triple.GET_PROPERTY() AS property
  FROM articles_rdf_data a WHERE a.id = 1;

PROPERTY
--------------------------------------------------------------------------------
<http://purl.org/dc/elements/1.1/title>

SELECT a.triple.GET_OBJECT() AS object
```

```
  FROM articles_rdf_data a WHERE a.id = 1;

OBJECT
--------------------------------------------------------------------------------
<All about XYZ>
```

1.5.1 Constructors for Inserting Triples Without Any Blank Nodes

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
  RETURN     SELF;

SDO_RDF_TRIPLE_S (
  model_name VARCHAR2, -- Model name
  subject    VARCHAR2, -- Subject
  property   VARCHAR2, -- Property
  object     CLOB) -- Object
  RETURN SELF;

GET_OBJ_VALUE() RETURN VARCHAR2;
```

Example 1–6 uses the first constructor format to insert a triple.

Example 1–6 SDO_RDF_TRIPLE_S Constructor to Insert a Triple

```
INSERT INTO articles_rdf_data VALUES (2,
  SDO_RDF_TRIPLE_S ('articles','<http://nature.example.com/Article1>',
    '<http://purl.org/dc/elements/1.1/creator>',
    '"Jane Smith"'));
```

1.5.2 Constructors for Inserting Triples With or Without Any Blank Nodes

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

```
SDO_RDF_TRIPLE_S (
  model_name VARCHAR2, -- Model name
  sub_or_bn  VARCHAR2, -- Subject or blank node
  property   VARCHAR2, -- Property
  obj_or_bn  VARCHAR2, -- Object or blank node
  bn_m_id    NUMBER) -- ID of the model from which to reuse the blank node
  RETURN SELF;

SDO_RDF_TRIPLE_S (
  model_name VARCHAR2, -- Model name
  sub_or_bn  VARCHAR2, -- Subject or blank node
  property   VARCHAR2, -- Property
  object     CLOB, -- Object
  bn_m_id    NUMBER) -- ID of the model from which to reuse the blank node
  RETURN SELF;
```

If the value of bn_m_id is positive, it must be the same as the model ID of the target model.

Example 1–7 uses the first constructor format to insert a triple that reuses a blank node for the subject.

Example 1–7 SDO_RDF_TRIPLE_S Constructor to Reusing a Blank Node

```
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>',
  4));
```

1.6 Using the SEM_MATCH Table Function to Query Semantic Data

To query semantic data, use the SEM_MATCH table function. This function has the following attributes:

```
SEM_MATCH(
  query        VARCHAR2,
  models       SEM_MODELS,
  rulebases    SEM_RULEBASES,
  aliases      SEM_ALIASES,
  filter       VARCHAR2,
  index_status VARCHAR2,
  options      VARCHAR2
 ) RETURN ANYDATASET;
```

The query attribute is 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 enclosed in parentheses. 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:

```
'(?x :grandParentOf ?y) (?x rdf:type :Male) (?y :height ?h)'
```

The models attribute identifies the model or models to use. Its data type is SEM_MODELS, which has the following definition: TABLE OF VARCHAR2(25). If you are querying a virtual model, specify only the name of the virtual model and no other models. (Virtual models are explained in Section 1.3.8.)

The rulebases attribute identifies one or more rulebases whose rules are to be applied to the query. Its data type is SDO_RDF_RULEBASES, which has the following definition: TABLE OF VARCHAR2(25). If you are querying a virtual model, 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:

```
('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).

The `index_status` attribute lets you query semantic data even when the relevant rules index does not have a valid status. (If you are querying a virtual model, this attribute refers to the rules index associated with the virtual model.) If this attribute is null, the query returns an error if the rules index 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 Section 1.6.1.

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

- `ALLOW_DUP=T` generates an underlying SQL statement that performs a "union all" instead of a union of the semantic models 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 models 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 semantic models are involved in a SEM_MATCH query.

  If you are querying a virtual model, specifying `ALLOW_DUP=T` causes the SEMV_*vm_name* view to be queried; otherwise, the SEMU_*vm_name* view is queried.

- `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 RDF model or models and optionally the corresponding rules index, 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*>). Hints used for influencing the results of queries include GET_CANON_VALUE(<*set of aliases for variables*>), which ensures that the values returned for the referenced variables will be their canonical lexical forms. These hints have the same format and basic meaning as hints in SQL statements, which are explained in *Oracle Database SQL Language Reference*.

Example 1–9 shows the HINT0 option used in a SEM_MATCH query.

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* that may be a literal (that is, for each variable that appears only in the object position in the query pattern), the result elements have the following attributes: *var*, *var*$RDFVID, *var*$_PREFIX, *var*$_SUFFIX, *var*$RDFVTYP, *var*$RDFCLOB, *var*$RDFLTYP, *var*$RDFVID, *var*$_PREFIX, *var*$_SUFFIX, and *var*$RDFLANG.
- For each variable *var* that cannot take a literal value, the result elements have the following attributes: *var* and *var*$RDFVTYP.

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 and 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.

Example 1–8 selects all grandfathers (grandparents who are male) and their grandchildren from the `family` model, using inferencing from both the `RDFS` and `family_rb` rulebases. (This example is an excerpt from Example 1–14 in Section 1.10.2.)

Example 1–8 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));
```

Example 1–9 is functionally the same as Example 1–8, but it adds the `HINT0` option.

Example 1–9 HINT0 Option with 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,
    'HINT0={LEADING(t0 t1) USE_NL(?x ?y) GET_CANON_VALUE(?x ?y)}'));
```

Example 1–10 uses the Pathway/Genome `BioPax` ontology to get all chemical compound types that belong to both `Proteins` and `Complexes`:

Example 1–10 SEM_MATCH Table Function

```
SELECT t.r
  FROM TABLE (SEM_MATCH (
      '(?r rdfs:subClassOf Proteins)
       (?r rdfs:subClassOf Complexes)',
      SEM_Models ('BioPax'),
      SEM_Rulebases ('rdfs'), NULL)) t;
```

As shown in Example 1–10, the search pattern for the SEM_MATCH table function is specified using SPARQL-like syntax where the variable starts with the question-mark character (`?`). In this example, the variable `?r` must match to the same term, and thus is must be a subclass of both `Proteins` and `Complexes`.

To use the SEM_RELATED operator to query an OWL ontology, see Section 2.3.

When you are querying multiple models or querying one or more models and the corresponding rules index, consider using virtual models (explained in Section 1.3.8) because of the potential performance benefits.

1.6.1 Performing Queries with Incomplete or Invalid Rules Indexes

You can query semantic data even when the relevant rules index 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 rules index status is stored in the STATUS column of the MDSYS.SEM_RULES_INDEX_INFO view, which is described in Section 1.3.7. The SEM_MATCH table function is described in Section 1.6.)

The index_status attribute value affects the query behavior as follows:

- If the rules index 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 rules index 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 rules index 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 rules index (invalid, incomplete, or valid).

However, the following considerations apply if the status of the rules index is incomplete or invalid:

- If the status is incomplete, the content of a rules index may be approximate, because some triples that are inferable (due to the recent insertions into the underlying models) may not actually be present in the rules index, and therefore results returned by the query may be inaccurate.
- If the status is invalid, the content of the rules index may be approximate, because some triples that are no longer inferable (due to recent modifications to the underlying models or rulebases, or both) may still be present in the rules index, 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 rules index.

1.6.2 Graph Patterns: Support for Curly Brace Syntax and OPTIONAL Keyword

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 keyword. With this syntax, you can also use the OPTIONAL construct to retrieve results even in the case of a partial match.

Example 1–11 is functionally the same as Example 1–8, but it uses the syntax with curly braces and a period to express a graph pattern in the SEM_MATCH table function.

Example 1–11 Curly Brace Syntax

```
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));
```

Example 1–12 uses the OPTIONAL construct to modify Example 1–11, so that also returns, for each grandfather, the names of the games that he plays or null if he does not play any games.

Example 1–12 Curly Brace Syntax and OPTIONAL Construct

```
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)}'));
```

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 using HINT0={<*hint-string*>} (explained in Section 1.6), should be constructed only on the basis of the portion of the graph pattern outside the

OPTIONAL construct. For example, the only valid aliases for use in a hint specification for the query in Example 1–12 are `t0`, `t1`, `?x`, and `?y`.

1.7 Loading and Exporting Semantic Data

To load semantic data into a model, use one or more of the following options:

- Bulk load using a SQL*Loader direct-path load to get data from an N-Triple format into a staging table and then use a PL/SQL procedure to load or append data into the semantic data store, as explained in Section 1.7.1.

  This is the fastest option for loading large amounts of data; however, it cannot handle triples containing object values with more than 4000 bytes.
- Batch load using a Java client interface to load or append data from an N-Triple format file into the semantic data store (see Section 1.7.2).

  This option is slower than bulk loading, but it handles triples containing object values with more than 4000 bytes.
- Load into tables using SQL INSERT statements that call the SDO_RDF_TRIPLE_S constructor, as explained in Section 1.7.3.

To export semantic data, use the Java API, as described in Section 1.7.4.

1.7.1 Bulk Loading Semantic Data Using a Staging Table

You can load semantic data (and optionally associated non-semantic data) in bulk using a staging table. The data must first be parsed to check for syntax correctness and then loaded into the staging table. Then, you can call the SEM_APIS.BULK_LOAD_FROM_STAGING_TABLE procedure (described in Chapter 3).

The following example shows the format for the staging table, including all required columns and the required names for these columns:

```
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
);
```

If you also want to load non-semantic data, specify additional columns for the non-semantic data in the CREATE TABLE statement. The non-semantic 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-semantic 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
);
```

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.

You must grant the following privileges to the MDSYS user: SELECT privilege on the staging table, and INSERT privilege on the application table.

You can use the SQL*Loader utility to parse and load semantic 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.

Objects longer than 4000 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, triples containing object values longer than 4000 bytes can be loaded using the following approach:

1. Use the SEM_APIS.BULK_LOAD_FROM_STAGING_TABLE procedure to load all rows that can be stored in the staging table.
2. Load the remaining rows (that is, the rejected rows when using SQL*Loader with the sample control file) from an N-Triple format file, as described in Section 1.7.2.

1.7.1.1 Recording Event Traces during Bulk Loading

If a table named RDF$ET_TAB exists in the invoker's schema and if the MDSYS user has been granted the INSERT and UPDATE privileges on this table, event traces for some of the tasks performed during executions of the SEM_APIS.BULK_LOAD_FROM_STAGING_TABLE procedure will be added to the table. You may find the content of this table useful if you ever need to report any problems in bulk load. The RDF$ET_TAB table must be created as follows:

```
CREATE TABLE RDF$ET_TAB (
  proc_sid VARCHAR2(30),
  proc_sig VARCHAR2(200),
  event_name varchar2(200),
  start_time timestamp,
  end_time timestamp,
  start_comment varchar2(1000) DEFAULT NULL,
  end_comment varchar2(1000) DEFAULT NULL
);
GRANT INSERT, UPDATE on RDF$ET_TAB to MDSYS;
```

1.7.2 Batch Loading Semantic Data Using the Java API

You can perform a batch load operation using the Java class `oracle.spatial.rdf.client.BatchLoader`, which is packaged in `<`*`ORACLE_HOME`*`>/md/jlib/sdordf.jar`. Before performing a batch load operation, ensure that the following are true:

- The semantic data is in N-Triple format. (Several tools are available for converting RDF/XML to N-Triple format; see the Oracle Technology Network or perform a Web search for information about RDF/XML to N-Triple conversion.)
- Oracle Database Release 11, with Oracle Spatial, is installed, and partitioning is enabled.
- A semantic technologies network, an application table, and its corresponding semantic model have been created in the database.
- The CLASSPATH definition includes `ojdbc5.jar`.

- You are using JDK version 1.5 or later. (You can use the Java version packaged under *<ORACLE_HOME>*/jdk/bin.)

To run the `oracle.spatial.rdf.client.BatchLoader` class, use a command (on a single command line) in the following general form (replacing the sample example database connection information with your own connection information).

- Linux systems:

```
java -Ddb.user=scott -Ddb.password=password -Ddb.host=127.0.0.1 -Ddb.port=1522
-Ddb.sid=orcl -classpath ${ORACLE_HOME}/md/jlib/sdordf.jar:${ORACLE_
HOME}/jdbc/lib/ojdbc5.jar oracle.spatial.rdf.client.BatchLoader <N-TripleFile>
<tablename> <tablespaceName> <modelName>
```

- Windows systems:

```
java -Ddb.user=scott -Ddb.password=password -Ddb.host=127.0.0.1 -Ddb.port=1522
-Ddb.sid=orcl -classpath %ORACLE_HOME%\md\jlib\sdordf.jar;%ORACLE_
HOME%\jdbc\lib\ojdbc5.jar oracle.spatial.rdf.client.BatchLoader <N-TripleFile>
<tablename> <tablespaceName> <modelName>
```

By default, `BatchLoader` assumes there are at least two columns, a column named ID of type NUMBER and a column named TRIPLE of type SDO_RDF_TRIPLE_S, in the user's application table. However, you can override the default names by using the JVM properties `-DidColumn=<idColumnName>` and `-DtripleColumn=<tripleColumnName>`. Note that the ID column is not required; and to prevent `BatchLoader` from generating a sequence-like identifier in the ID column for each triple inserted, specify the JVM property `-DjustTriple=true`.

If the application table is not empty and if you want the batch loading to be done in append mode, specify an additional JVM property: `-Dappend=true`. Moreover, in append mode you might want to choose a different starting value for ID column in user's application table, and to accomplish this you can add the JVM property `-DstartID=<startingIntegerValue>` to the command line. By default, the ID column starts at 1 and is increased sequentially as new triples are inserted into the application table.

To skip the first *n* triples in `<N-TripleFile>`, add the JVM property `-Dskip=<numberOfTriplesSkipped>` to the command line.

To load an N-Triple file with a character set different from the default, specify the JVM property `-Dcharset=<charsetName>`. For example, `-Dcharset="UTF-8"` will recognize UTF-8 encoding. However, for UTF-8 characters to be stored properly in the N-Triple file, the Oracle database must be configured to use a corresponding universal character set, such as AL32UTF8.

The `BatchLoader` class supports loading an N-Triple file in compressed format. If the `<N-TripleFile>` has a file extension of .zip or .jar, the file will be uncompressed and loaded at the same time.

1.7.2.1 When to Choose Batch Loading

Batch loading is faster than loading semantic data using INSERT statements (described in Section 1.7.3). However, bulk loading (described in Section 1.7.1) is much faster than batch loading for large amounts of data. Batch loading is typically a good option when the following conditions are true:

- The data to be loaded is less than a few million triples.
- The data contains a significant amount long literals (longer than 4000 bytes).

1.7.3 Loading Semantic Data Using INSERT Statements

To load semantic 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 Section 1.5.1.

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 includes statements with URIs, a blank node (the model_id for nsu is 4), 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"'));
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>', 4));
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'));
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'));
```

To convert semantic XML data to INSERT statements, you can edit the sample `rss2insert.xsl` XSLT file to convert all the features in the semantic data XML file. The blank node constructor is used to insert statements with blank nodes. After editing the XSLT, download the Xalan XSLT processor (`http://xml.apache.org/xalan`) and follow the installation instructions. To convert a semantic data XML file to INSERT statements using your edited version of the `rss2insert.xsl` file, use a command in the following format:

```
java org.apache.xalan.xslt.Process -in input.rdf -xsl rss2insert.xsl -out
output.nt
```

1.7.4 Exporting Semantic Data

To output semantic data to a file in N-Triple format, use the `NTripleConverter` Java class. The `NDM2NTriple(String, int)` method exports all the triples stored for the specified model.

For information about using the `NTriple converter` class, see the `README.txt` file in the `sdordf_converter.zip` file, which you can download from the Oracle Technology Network.

1.8 Creating and Managing Semantic Network Indexes

Semantic network indexes are nonunique B-tree indexes that you can add, alter, and drop for use with models and rules indexes in a semantic network. You can use such indexes to tune the performance of SEM_MATCH queries on the models and rules indexes in the network. You can create and manage semantic network indexes using the following subprograms:

- SEM_APIS.ADD_SEM_INDEX
- SEM_APIS.ALTER_SEM_INDEX_ON_MODEL

- SEM_APIS.ALTER_SEM_INDEX_ON_RULES_INDEX
- SEM_APIS.DROP_SEM_INDEX

All of these subprograms have an index_code parameter, which can contain any sequence of the following letters (without repetition): `P`, `C`, `S`, `M`, `O`. 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, MODEL_ID, and END_NODE_ID.

The SEM_APIS.ADD_SEM_INDEX procedure creates a semantic network index that results in creation of a nonunique B-tree index in UNUSABLE status for each of the existing models and rules indexes. The name of the index is RDF_LNK_<*index_code*>_IDX and the index is owned by MDSYS. This operation is allowed only if the invoker has DBA role. The following example shows creation of the `PSCM` index with the following key: <P_VALUE_ID, START_NODE_ID, CANON_END_NODE_ID, MODEL_ID>.

```
EXECUTE SEM_APIS.ADD_SEM_INDEX('PSCM');
```

After you create a semantic 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 semantic network index usable or unusable for specific models or rules indexes that you own by calling the SEM_APIS.ALTER_SEM_INDEX_ON_MODEL and SEM_APIS.ALTER_SEM_INDEX_ON_RULES_INDEX procedures and specifying `'REBUILD'` or `'UNUSABLE'` as the `command` parameter. Thus, you can experiment by making different semantic network indexes usable and unusable, and checking for any differences in performance. For example, the following statement makes the `PSCM` index usable for the `FAMILY` model:

```
EXECUTE SEM_APIS.ALTER_SEM_INDEX_ON_MODEL('FAMILY','PSCM','REBUILD');
```

Also note the following:

- Independent of any semantic network indexes that you create, when a semantic 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 `'PSCF'` when you call the subprograms mentioned in this section.
- When you create a new model or a new rules index, a new nonunique B-tree index is created for each of the semantic network indexes, and each such B-tree index is in the USABLE status.
- Including the MODEL_ID column in a semantic network index key (by including 'M' in the `index_code` value) may improve query performance. This is particularly relevant when virtual models are used.

1.9 Quick Start for Using Semantic Data

To work with semantic data in an Oracle database, follow these general steps:

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 '/oradata/orcl/rdf_tblspace.dat' SIZE 1024M REUSE
 AUTOEXTEND ON NEXT 256M MAXSIZE UNLIMITED
 SEGMENT SPACE MANAGEMENT AUTO;
```

2. Create a semantic data network.

   Creating a semantic data network adds semantic data support to an Oracle database. You must create a semantic data network as a user with DBA privileges, specifying a valid tablespace with adequate space. Create the network only once for an Oracle database.

   The following example creates a semantic data network using a tablespace named RDF_TBLSPACE (which must already exist):

   ```
   EXECUTE SEM_APIS.CREATE_SEM_NETWORK('rdf_tblspace');
   ```

3. Connect as the database user under whose schema you will store your semantic data; do not perform the following steps while connected as SYS, SYSTEM, or MDSYS.

4. Create a table to store references to the semantic data. (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:

   ```
   CREATE TABLE articles_rdf_data (id NUMBER, triple SDO_RDF_TRIPLE_S);
   ```

5. Create a model.

   When you create a model, you 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, which will use the table created in the preceding step.

   ```
   EXECUTE SEM_APIS.CREATE_SEM_MODEL('articles', 'articles_rdf_data', 'triple');
   ```

6. Where possible, create Oracle database indexes on conditions that will be specified in the WHERE clause of SELECT statements, to provide better performance for direct queries against the application table's SDO_RDF_TRIPLE_S column. (These indexes are not relevant if the SEM_MATCH table function is being used.) The following example creates such indexes:

   ```
   -- Create indexes on the subjects, properties, and objects
   -- in the ARTICLES_RDF_DATA table.
   CREATE INDEX articles_sub_idx ON articles_rdf_data (triple.GET_SUBJECT());
   CREATE INDEX articles_prop_idx ON articles_rdf_data (triple.GET_PROPERTY());
   CREATE INDEX articles_obj_idx ON articles_rdf_data (TO_CHAR(triple.GET_
   OBJECT()));
   ```

After you create the model, you can insert triples into the table, as shown in the examples in Section 1.10.

1.10 Semantic Data Examples

This section contains the following PL/SQL examples:

- Section 1.10.1, "Example: Journal Article Information"
- Section 1.10.2, "Example: Family Information"

1.10.1 Example: Journal Article Information

This section presents a simplified PL/SQL example of model for statements about journal articles. Example 1–13 contains descriptive comments, refer to concepts that are explained in this chapter, and uses functions and procedures documented in Chapter 3.

Example 1–13 Using a Model for Journal Article Information

```
-- Basic steps:
-- After you have connected as a privileged user and called
-- SEM_APIS.CREATE_RDF_NETWORK to add RDF support,
-- connect as a regular database user and do the following.
-- 1. For each desired model, create a table to hold its data.
-- 2. For each model, create a model (SEM_APIS.CREATE_RDF_MODEL).
-- 3. For each table to hold semantic data, insert data into the table.
-- 4. Use various subprograms and consructors.

-- Create the table to hold data for the model.
CREATE TABLE articles_rdf_data (id NUMBER, triple SDO_RDF_TRIPLE_S);

-- Create the model.
EXECUTE SEM_APIS.CREATE_RDF_MODEL('articles', 'articles_rdf_data', 'triple');

-- 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.

-- Insert rows into the table.

-- Article1 has the title "All about XYZ".
INSERT INTO articles_rdf_data VALUES (1,
  SDO_RDF_TRIPLE_S ('articles','http://nature.example.com/Article1',
    'http://purl.org/dc/elements/1.1/title','All about XYZ'));

-- Article1 was created (written) by Jane Smith.
INSERT INTO articles_rdf_data VALUES (2,
  SDO_RDF_TRIPLE_S ('articles','http://nature.example.com/Article1',
    'http://purl.org/dc/elements/1.1/creator',
    'Jane Smith'));

-- Article1 references (refers to) Article2.
INSERT INTO articles_rdf_data VALUES (3,
  SDO_RDF_TRIPLE_S ('articles',
    'http://nature.example.com/Article1',
    'http://purl.org/dc/terms/references',
    'http://nature.example.com/Article2'));

-- Article1 references (refers to) Article3.
INSERT INTO articles_rdf_data VALUES (4,
  SDO_RDF_TRIPLE_S ('articles',
    'http://nature.example.com/Article1',
    'http://purl.org/dc/terms/references',
```

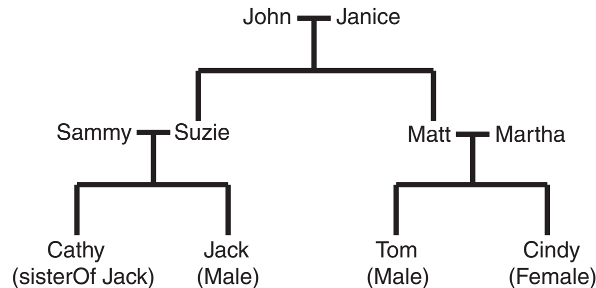
```
    'http://nature.example.com/Article3'));

-- Article2 has the title "A review of ABC".
INSERT INTO articles_rdf_data VALUES (5,
  SDO_RDF_TRIPLE_S ('articles',
    'http://nature.example.com/Article2',
    'http://purl.org/dc/elements/1.1/title',
    'A review of ABC'));

-- Article2 was created (written) by Joe Bloggs.
INSERT INTO articles_rdf_data VALUES (6,
  SDO_RDF_TRIPLE_S ('articles',
    'http://nature.example.com/Article2',
    'http://purl.org/dc/elements/1.1/creator',
    'Joe Bloggs'));

-- Article2 references (refers to) Article3.
INSERT INTO articles_rdf_data VALUES (7,
  SDO_RDF_TRIPLE_S ('articles',
    'http://nature.example.com/Article2',
    'http://purl.org/dc/terms/references',
    'http://nature.example.com/Article3'));

COMMIT;

-- Query semantic data.

SELECT SEM_APIS.GET_MODEL_ID('articles') AS model_id FROM DUAL;

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;

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;

-- Use SDO_RDF_TRIPLE_S member functions in queries.

SELECT a.triple.GET_TRIPLE() AS triple
  FROM articles_rdf_data a WHERE a.id = 1;
SELECT a.triple.GET_SUBJECT() AS subject
  FROM articles_rdf_data a WHERE a.id = 1;
SELECT a.triple.GET_PROPERTY() AS property
  FROM articles_rdf_data a WHERE a.id = 1;
SELECT a.triple.GET_OBJECT() AS object
  FROM articles_rdf_data a WHERE a.id = 1;
```

1.10.2 Example: Family Information

This section presents a simplified PL/SQL example of a model for statements about family tree (genealogy) information. Example 1–13 contains descriptive comments, refer to concepts that are explained in this chapter, and uses functions and procedures documented in Chapter 3.

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 1–14 Using a Model for Family Information

```
-- Basic steps:
-- After you have connected as a privileged user and called
-- SEM_APIS.CREATE_RDF_NETWORK to enable RDF support,
-- connect as a regular database user and do the following.
-- 1. For each desired model, create a table to hold its data.
-- 2. For each model, create a model (SEM_APIS.CREATE_RDF_MODEL).
-- 3. For each table to hold semantic data, insert data into the table.
-- 4. Use various subprograms and constructors.

-- Create the table to hold data for the model.
CREATE TABLE family_rdf_data (id NUMBER, triple SDO_RDF_TRIPLE_S);

-- Create the model.
execute SEM_APIS.create_rdf_model('family', 'family_rdf_data', 'triple');

-- Insert rows into the table. These express the following information:
-----------------
-- John and Janice have two children, Suzie and Matt.
-- Matt married Martha, and they have two children:
--   Tom (male, height 5.75) and Cindy (female, height 06.00).
-- Suzie married Sammy, and they have two children:
--   Cathy (height 5.8) and Jack (male, height 6).

-- 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.
------------------------

-- John is the father of Suzie.
INSERT INTO family_rdf_data VALUES (1,
SDO_RDF_TRIPLE_S('family',
'http://www.example.org/family/John',
'http://www.example.org/family/fatherOf',
'http://www.example.org/family/Suzie'));

-- John is the father of Matt.
INSERT INTO family_rdf_data VALUES (2,
SDO_RDF_TRIPLE_S('family',
```

```
'http://www.example.org/family/John',
'http://www.example.org/family/fatherOf',
'http://www.example.org/family/Matt'));

-- Janice is the mother of Suzie.
INSERT INTO family_rdf_data VALUES (3,
SDO_RDF_TRIPLE_S('family',
'http://www.example.org/family/Janice',
'http://www.example.org/family/motherOf',
'http://www.example.org/family/Suzie'));

-- Janice is the mother of Matt.
INSERT INTO family_rdf_data VALUES (4,
SDO_RDF_TRIPLE_S('family',
'http://www.example.org/family/Janice',
'http://www.example.org/family/motherOf',
'http://www.example.org/family/Matt'));

-- Sammy is the father of Cathy.
INSERT INTO family_rdf_data VALUES (5,
SDO_RDF_TRIPLE_S('family',
'http://www.example.org/family/Sammy',
'http://www.example.org/family/fatherOf',
'http://www.example.org/family/Cathy'));

-- Sammy is the father of Jack.
INSERT INTO family_rdf_data VALUES (6,
SDO_RDF_TRIPLE_S('family',
'http://www.example.org/family/Sammy',
'http://www.example.org/family/fatherOf',
'http://www.example.org/family/Jack'));

-- Suzie is the mother of Cathy.
INSERT INTO family_rdf_data VALUES (7,
SDO_RDF_TRIPLE_S('family',
'http://www.example.org/family/Suzie',
'http://www.example.org/family/motherOf',
'http://www.example.org/family/Cathy'));

-- Suzie is the mother of Jack.
INSERT INTO family_rdf_data VALUES (8,
SDO_RDF_TRIPLE_S('family',
'http://www.example.org/family/Suzie',
'http://www.example.org/family/motherOf',
'http://www.example.org/family/Jack'));

-- Matt is the father of Tom.
INSERT INTO family_rdf_data VALUES (9,
SDO_RDF_TRIPLE_S('family',
'http://www.example.org/family/Matt',
'http://www.example.org/family/fatherOf',
'http://www.example.org/family/Tom'));

-- Matt is the father of Cindy
INSERT INTO family_rdf_data VALUES (10,
SDO_RDF_TRIPLE_S('family',
'http://www.example.org/family/Matt',
'http://www.example.org/family/fatherOf',
'http://www.example.org/family/Cindy'));
```

```
-- Martha is the mother of Tom.
INSERT INTO family_rdf_data VALUES (11,
SDO_RDF_TRIPLE_S('family',
'http://www.example.org/family/Martha',
'http://www.example.org/family/motherOf',
'http://www.example.org/family/Tom'));

-- Martha is the mother of Cindy.
INSERT INTO family_rdf_data VALUES (12,
SDO_RDF_TRIPLE_S('family',
'http://www.example.org/family/Martha',
'http://www.example.org/family/motherOf',
'http://www.example.org/family/Cindy'));

-- Cathy is the sister of Jack.
INSERT INTO family_rdf_data VALUES (13,
SDO_RDF_TRIPLE_S('family',
'http://www.example.org/family/Cathy',
'http://www.example.org/family/sisterOf',
'http://www.example.org/family/Jack'));

-- Jack is male.
INSERT INTO family_rdf_data VALUES (14,
SDO_RDF_TRIPLE_S('family',
'http://www.example.org/family/Jack',
'http://www.w3.org/1999/02/22-rdf-syntax-ns#type',
'http://www.example.org/family/Male'));

-- Tom is male.
INSERT INTO family_rdf_data VALUES (15,
SDO_RDF_TRIPLE_S('family',
'http://www.example.org/family/Tom',
'http://www.w3.org/1999/02/22-rdf-syntax-ns#type',
'http://www.example.org/family/Male'));

-- Cindy is female.
INSERT INTO family_rdf_data VALUES (16,
SDO_RDF_TRIPLE_S('family',
'http://www.example.org/family/Cindy',
'http://www.w3.org/1999/02/22-rdf-syntax-ns#type',
'http://www.example.org/family/Female'));

-- Person is a class.
INSERT INTO family_rdf_data VALUES (17,
SDO_RDF_TRIPLE_S('family',
'http://www.example.org/family/Person',
'http://www.w3.org/1999/02/22-rdf-syntax-ns#type',
'http://www.w3.org/2000/01/rdf-schema#Class'));

-- Male is a subclass of Person.
INSERT INTO family_rdf_data VALUES (18,
SDO_RDF_TRIPLE_S('family',
'http://www.example.org/family/Male',
'http://www.w3.org/2000/01/rdf-schema#subClassOf',
'http://www.example.org/family/Person'));

-- Female is a subclass of Person.
INSERT INTO family_rdf_data VALUES (19,
SDO_RDF_TRIPLE_S('family',
'http://www.example.org/family/Female',
```

```
'http://www.w3.org/2000/01/rdf-schema#subClassOf',
'http://www.example.org/family/Person'));

-- siblingOf is a property.
INSERT INTO family_rdf_data VALUES (20,
SDO_RDF_TRIPLE_S('family',
'http://www.example.org/family/siblingOf',
'http://www.w3.org/1999/02/22-rdf-syntax-ns#type',
'http://www.w3.org/1999/02/22-rdf-syntax-ns#Property'));

-- parentOf is a property.
INSERT INTO family_rdf_data VALUES (21,
SDO_RDF_TRIPLE_S('family',
'http://www.example.org/family/parentOf',
'http://www.w3.org/1999/02/22-rdf-syntax-ns#type',
'http://www.w3.org/1999/02/22-rdf-syntax-ns#Property'));

-- brotherOf is a subproperty of siblingOf.
INSERT INTO family_rdf_data VALUES (22,
SDO_RDF_TRIPLE_S('family',
'http://www.example.org/family/brotherOf',
'http://www.w3.org/2000/01/rdf-schema#subPropertyOf',
'http://www.example.org/family/siblingOf'));

-- sisterOf is a subproperty of siblingOf.
INSERT INTO family_rdf_data VALUES (23,
SDO_RDF_TRIPLE_S('family',
'http://www.example.org/family/sisterOf',
'http://www.w3.org/2000/01/rdf-schema#subPropertyOf',
'http://www.example.org/family/siblingOf'));

-- A brother is male.
INSERT INTO family_rdf_data VALUES (24,
SDO_RDF_TRIPLE_S('family',
'http://www.example.org/family/brotherOf',
'http://www.w3.org/2000/01/rdf-schema#domain',
'http://www.example.org/family/Male'));

-- A sister is female.
INSERT INTO family_rdf_data VALUES (25,
SDO_RDF_TRIPLE_S('family',
'http://www.example.org/family/sisterOf',
'http://www.w3.org/2000/01/rdf-schema#domain',
'http://www.example.org/family/Female'));

-- fatherOf is a subproperty of parentOf.
INSERT INTO family_rdf_data VALUES (26,
SDO_RDF_TRIPLE_S('family',
'http://www.example.org/family/fatherOf',
'http://www.w3.org/2000/01/rdf-schema#subPropertyOf',
'http://www.example.org/family/parentOf'));

-- motherOf is a subproperty of parentOf.
INSERT INTO family_rdf_data VALUES (27,
SDO_RDF_TRIPLE_S('family',
'http://www.example.org/family/motherOf',
'http://www.w3.org/2000/01/rdf-schema#subPropertyOf',
'http://www.example.org/family/parentOf'));

-- A father is male.
```

```
INSERT INTO family_rdf_data VALUES (28,
SDO_RDF_TRIPLE_S('family',
'http://www.example.org/family/fatherOf',
'http://www.w3.org/2000/01/rdf-schema#domain',
'http://www.example.org/family/Male'));

-- A mother is female.
INSERT INTO family_rdf_data VALUES (29,
SDO_RDF_TRIPLE_S('family',
'http://www.example.org/family/motherOf',
'http://www.w3.org/2000/01/rdf-schema#domain',
'http://www.example.org/family/Female'));

-- Use SET ESCAPE OFF to prevent the caret (^) from being
-- interpreted as an escape character. Two carets (^^) are
-- used to represent typed literals.
SET ESCAPE OFF;

-- Cathy's height is 5.8 (decimal).
INSERT INTO family_rdf_data VALUES (30,
SDO_RDF_TRIPLE_S('family',
'http://www.example.org/family/Cathy',
'http://www.example.org/family/height',
'"5.8"^^xsd:decimal'));

-- Jack's height is 6 (integer).
INSERT INTO family_rdf_data VALUES (31,
SDO_RDF_TRIPLE_S('family',
'http://www.example.org/family/Jack',
'http://www.example.org/family/height',
'"6"^^xsd:integer'));

-- Tom's height is 05.75 (decimal).
INSERT INTO family_rdf_data VALUES (32,
SDO_RDF_TRIPLE_S('family',
'http://www.example.org/family/Tom',
'http://www.example.org/family/height',
'"05.75"^^xsd:decimal'));

-- Cindy's height is 06.00 (decimal).
INSERT INTO family_rdf_data VALUES (33,
SDO_RDF_TRIPLE_S('family',
'http://www.example.org/family/Cindy',
'http://www.example.org/family/height',
'"06.00"^^xsd:decimal'));

COMMIT;

-- RDFS inferencing in the family model
BEGIN
  SEM_APIS.CREATE_RULES_INDEX(
    'rdfs_rix_family',
    SEM_Models('family'),
    SEM_Rulebases('RDFS'));
END;
/

-- Select all males from the family model, without inferencing.
SELECT m
  FROM TABLE(SEM_MATCH(
```

```
    '(?m rdf:type :Male)',
    SEM_Models('family'),
    null,
    SEM_ALIASES(SEM_ALIAS('','http://www.example.org/family/')),
    null));

-- Select all males from the family model, with RDFS inferencing.
SELECT m
  FROM TABLE(SEM_MATCH(
    '(?m rdf:type :Male)',
    SEM_Models('family'),
    SDO_RDF_Rulebases('RDFS'),
    SEM_ALIASES(SEM_ALIAS('','http://www.example.org/family/')),
    null));

-- General inferencing in the family model

EXECUTE SEM_APIS.CREATE_RULEBASE('family_rb');

INSERT INTO mdsys.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
-- rules index, drop the preceding rules index and then re-create it.
EXECUTE SEM_APIS.DROP_RULES_INDEX ('rdfs_rix_family');

-- Re-create the rules index.
BEGIN
  SEM_APIS.CREATE_RULES_INDEX(
    'rdfs_rix_family',
    SEM_Models('family'),
    SEM_Rulebases('RDFS','family_rb'));
END;
/

-- Select all grandfathers and their grandchildren from the family model,
-- without inferencing. (With no inferencing, no results are returned.)
SELECT x grandfather, y grandchild
  FROM TABLE(SEM_MATCH(
    '(?x :grandParentOf ?y) (?x rdf:type :Male)',
    SEM_Models('family'),
    null,
    SEM_ALIASES(SEM_ALIAS('','http://www.example.org/family/')),
    null));

-- Select all grandfathers and their grandchildren from the family model.
-- Use inferencing from both the RDFS and family_rb rulebases.
SELECT x grandfather, y grandchild
  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));
```

```
-- Set up to find grandfathers of tall (>= 6) grandchildren
-- from the family model, with RDFS inferencing and
-- inferencing using the "family_rb" rulebase.

UPDATE mdsys.semr_family_rb SET
  antecedents = '(?x :parentOf ?y) (?y :parentOf ?z) (?z :height ?h)',
  filter = '(h >= 6)',
  aliases = SEM_ALIASES(SEM_ALIAS('','http://www.example.org/family/'))
WHERE rule_name = 'GRANDPARENT_RULE';

-- Because the rulebase has been updated, drop the preceding rules index,
-- and then re-create it.
EXECUTE SEM_APIS.DROP_RULES_INDEX ('rdfs_rix_family');

-- Re-create the rules index.
BEGIN
  SEM_APIS.CREATE_RULES_INDEX(
    'rdfs_rix_family',
    SEM_Models('family'),
    SEM_Rulebases('RDFS','family_rb'));
END;
/

-- Find the rules index that was just created (that is, the
-- one based on the specified model and rulebases).
SELECT SEM_APIS.LOOKUP_RULES_INDEX(SEM_MODELS('family'),
  SEM_RULEBASES('RDFS','family_rb')) AS lookup_rules_index FROM DUAL;

-- Select grandfathers of tall (>= 6) grandchildren, and their
-- tall grandchildren.
SELECT x grandfather, y grandchild
  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));
```

1.11 Required Procedure for Semantic Technologies Support

Before you can use any types, synonyms, or PL/SQL packages related to Oracle semantic technologies support, you must connect to the database as a user with DBA privileges, start SQL*Plus, and execute one of the following scripts, depending on your needs:

- Upgrade with data migration: `$ORACLE_HOME/md/admin/catsem10i.sql`
- New installation: `$ORACLE_HOME/md/admin/catsem11i.sql`

Note: If you are upgrading from Oracle Database Release 10.2 and have any Release 10.2 RDF data, and if you do *not* want the RDF data to be migrated, you must do the following *before* running the `catsem10i.sql` script:

- Drop the RDF network.
- Drop all columns of types SDO_RDF_TRIPLE_S and SDO_RDF_TRIPLE in user tables.

The `catsem10i.sql` script is comprehensive and handles both installation and (if necessary) any RDF data migration from Release 10.2 to Release 11 format. This script is required if you have an existing Release 10.2 RDF network; however, it may take a long time to run if the existing Release 10.2 RDF network contains large amount of RDF data.

The `catsem11i.sql` script is sufficient if you have a new Release 11.1 installation or if you are upgrading from Release 10.2 but do not have any existing RDF network and any columns of types SDO_RDF_TRIPLE_S or SDO_RDF_TRIPLE in user tables.

If a Release 10.2 RDF network exists or if dependencies exist on the Release 10.2 RDF types SDO_RDF_TRIPLE_S or SDO_RDF_TRIPLE, the `catsem11i.sql` script will exit with an error. In that case, you can continue using Release 10.2 RDF support and run the `catsem10i.sql` script later, when you are ready to upgrade.

1.12 Partitioning Requirement

Before you can use any of the RDF and OWL capabilities, the Partitioning option must be enabled.

For licensing information about the Partitioning option, see *Oracle Database Licensing Information*. For usage information about partitioning, see *Oracle Database VLDB and Partitioning Guide*.

1.13 Downgrading to the Previous Oracle Database Release

If you need to downgrade to Oracle Database Release 10.2, and if you used Oracle Database Release 11 RDF or OWL features and want to preserve existing semantic data and rulebases, you must execute a statement to prepare for the downgrade, perform the downgrade, and execute another statement to restore the semantic data.

However, the following considerations apply:

- Entailed graph data will not be preserved, because the same information can be regenerated using the Oracle Database Release 10.2 RDF inference API.
- No rulebases or rules indexes related to OWL are preserved, because Oracle Database Release 10.2 did not support the OWL vocabulary.

Perform the following steps:

1. Before the database downgrade, connect to the Release 11 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:

   ```
   EXECUTE SDO_SEM_DOWNGRADE.PREPARE_DOWNGRADE_FROM_11;
   ```

   When this statement executes successfully, all existing semantic data and rulebases are saved. You will restore the semantic data after the database downgrade.
3. Perform the database downgrade.
4. Download the following file from the Semantic Technologies page of the Oracle Technology Network site: `sdosemdgu.plb`
5. If (and only if) your Oracle Database Release 10.2 release number is 10.2.0.1, click the Software link, and download and install the RDF-specific patch. (This patch is

needed because Release 10.2.0.1 did not have the batch loading feature, which is used to restore the semantic data.)

6. Connect to the Release 10.2 database as the SYS user with SYSDBA privileges.
7. Start SQL*Plus and enter a statement in the following statement:

```
EXECUTE SDO_SEM_DOWNGRADE_UTL.PREPARE_DOWNGRADE_TO_102('<tablespace-name>');
```

Where <*tablespace-name*> is the name of the tablespace in which the RDF network will be created.

When this statement executes successfully, all semantic data that had been saved before the downgrade is restored and ready to use.

1.14 Software Naming Changes for Semantic Technologies

Because the support for semantic 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 1–12 lists the old and new names for some objects related to support for semantic technologies, in alphabetical order by old name.

Table 1–12 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
SDO_RDF_MATCH table function	SEM_MATCH
SDO_RDF_MODELS data type	SEM_MODELS
SDO_RDF_RULEBASES data type	SEM_RULEBASES

2 OWL Concepts

This chapter describes concepts related to the support for a subset of the Web Ontology Language (OWL). It builds on the information in Chapter 1, 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/`.

2.1 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.

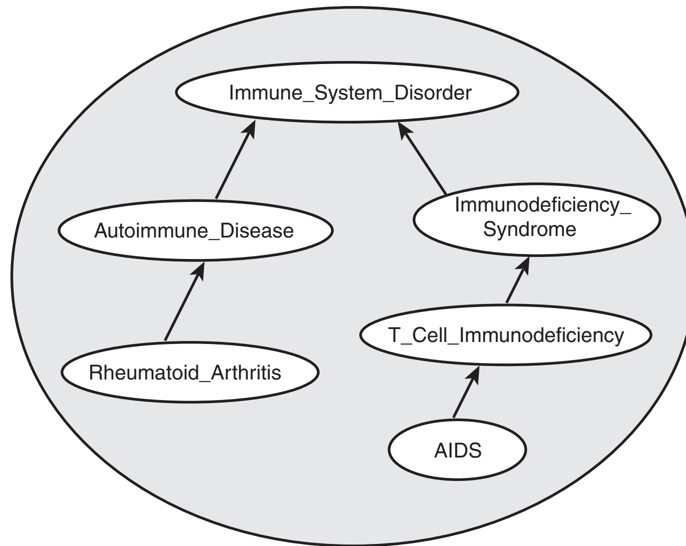
OWL ontologies build on RDFS-based ontologies by additionally allowing specification of property characteristics. OWL ontologies can be further classified as OWL-Lite, OWL-DL, and OWL Full. OWL-Lite restricts the cardinality minimum and maximum values to 0 or 1. OWL-DL relaxes this restriction by allowing arbitrary values for 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.

Section 2.1.2 describes OWL capabilities that are supported and not supported with semantic data.

2.1.1 Example: Cancer Ontology

Figure 2–1 shows part of a cancer ontology, which describes the classes and properties related to cancer. 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 2–1 Cancer Ontology Example



In the cancer ontology shown in Figure 2–1, 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 2–1.

Table 2–1 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 (described in Section 1.6) or the SEM_RELATED operator and its ancillary operators (described in Section 2.3).

2.1.2 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. Each supported vocabulary has a corresponding rulebase; however, these rulebases do not need to be populated because the underlying entailment 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).

The following OWL capabilities are not yet supported in any Oracle-supported OWL subset:

- Property restrictions: cardinality
- Class expressions: set operations (union, intersection), enumeration

Table 2–2 lists the RDFS/OWL vocabulary constructs included in each supported rulebase.

Table 2–2 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 2–2 (Cont.) RDFS/OWL Vocabulary Constructs Included in Each Supported

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

2.2 Using OWL Inferencing

You can use entailment rules to perform native OWL inferencing. This section creates a simple ontology, performs native inferencing, and illustrates some more advanced features.

2.2.1 Creating a Simple OWL Ontology

Example 2–1 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 2–1 Creating a Simple OWL Ontology

```
SQL> CREATE TABLE owltst(id number, triple sdo_rdf_triple_s);
Table created.

SQL> EXECUTE sem_apis.create_sem_model('owltst','owltst','triple');
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'));
1 row created.

SQL> commit;

SQL> -- Use SEM_MATCH to perform a simple query.
SQL> select s,p,o from table(SEM_MATCH('(?s ?p  ?o)', SEM_Models('OWLTST'),
           null,  null, null ));
```

2.2.2 Performing Native OWL inferencing

Example 2–2 calls the SEM_APIS.CREATE_ENTAILMENT procedure. You do not need to create the rulebase and add rules to it, because the OWL rules are already built into the Oracle semantic technologies inferencing engine.

Example 2–2 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_entailment('owltst_idx', sem_models('owltst'), sem_
rulebases('OWLPRIME'));
PL/SQL procedure successfully completed.

SQL> -- The following view is generated to represent the entailed graph (rules
index).
SQL> desc mdsys.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,p,o FROM table(SEM_MATCH('(?s ?p  ?o)', SEM_MODELS('OWLTST'),
           SEM_RULEBASES('OWLPRIME'),  null, null ));
```

2.2.3 Performing OWL and User-Defined Rules inferencing

Example 2–3 creates a user-defined rulebase, inserts a deliberately oversimplified `uncleOf` rule (stating that the brother of one's father is one's uncle), and calls the SEM_APIS.CREATE_ENTAILMENT procedure.

Example 2–3 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'));


SQL> INSERT INTO owltst VALUES (1, sdo_rdf_triple_s('owltst',
       'http://example.com/name/Jack', 'http://example.com/rel/botherOf',
       'http://example.com/name/John'));

SQL> -- Create a user-defined rulebase.

SQL> EXECUTE sem_apis.create_rulebase('user_rulebase');

SQL> -- Insert a simple "uncle" rule.

SQL> INSERT INTO mdsys.semr_user_rulebase VALUES ('uncle_rule',
'(?x <http://example.com/rel/botherOf> ?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_entailment('owltst2_idx', sem_models('owltst'),
          sem_rulebases('OWLPRIME','USER_RULEBASE'),
          SEM_APIS.REACH_CLOSURE, null, 'USER_RULES=T');
```

```
SQL> -- In the result of the following query, :Jack :uncleOf :Mary is inferred.
SQL> SELECT s,p,o FROM table(SEM_MATCH('(?s ?p  ?o)',
           SEM_MODELS('OWLTST'),
           SEM_RULEBASES('OWLPRIME','USER_RULEBASE'),  null, null ));
```

2.2.4 Generating OWL inferencing Proofs

OWL inferencing 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. The question arises, how can we trust inferred results? The answer involves using 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_ENTAILMENT procedure, as shown in the following example:

```
EXECUTE sem_apis.create_entailment('owltst_idx', sem_models('owltst'), -
  sem_rulebases('owlprime'), SEM_APIS.REACH_CLOSURE, 'SAM', 'PROOF=T');
```

Specifying `PROOF=T` causes a view to be created containing proof for each inferred triple. The view name is the entailment name prefixed by `MDSYS.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 mdsys.semi_owltst_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* ... *link-idn*] : *rule-name*

To get the full subject, predicate, and object URIs for proofs, you can query the model view and the entailment (rules index) view. Example 2–4 displays the LINK_ID value and associated triple contents using the model view MDSYS.SEMM_OWLTST and the entailment view MDSYS.SEMI_OWLTST_IDX.

Example 2–4 Displaying Proof Information

```
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 mdsys.semm_owltst union all
           select * from mdsys.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 2–4, 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`.

2.2.5 Validating OWL Models and Entailments

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_MODEL and SEM_APIS.VALIDATE_ENTAILMENT functions to detect inconsistencies in the original data model and in the entailment, respectively.

Example 2–5 shows uses the SEM_APIS.VALIDATE_ENTAILMENT function, which returns a null value if no errors are detected or a VARRAY of strings if any errors are detected.

Example 2–5 Validating an Entailment

```
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'));

SQL> -- Drop entailment first.
SQL>  exec sem_apis.drop_entailment('owltst_idx');
PL/SQL procedure successfully completed.

SQL> -- Perform OWL inferencing.
SQL> exec sem_apis.create_entailment('owltst_idx', sem_models('OWLTST'), sem_
rulebases('OWLPRIME'));
PL/SQL procedure successfully completed.

SQL > set serveroutput on;
SQL > -- Now invoke validation API: sem_apis.validate_entailment
SQL >
declare
  lva mdsys.rdf_longVarcharArray;
  idx int;
begin
  lva := sem_apis.validate_entailment(sem_models('OWLTST'), sem_
rulebases('OWLPRIME')) ;

  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 2–5)
- Error code (`10001` Example 2–5)
- One or more triples (shown in parentheses in the output; `(4_2_4_8 2 4 8)` in Example 2–5).

  These numbers are the LINK_ID value and the ID values of the subject, predicate, and object.
- Descriptive error message (`Unsatisfiable class.` in Example 2–5)

The output in Example 2–5 indicates that the error is caused by one triple that asserts that a class is a subclass of an empty class `owl:Nothing`.

2.2.6 Using SEM_APIS.CREATE_ENTAILMENT for RDFS Inference

In addition to accepting OWL vocabularies, the SEM_APIS.CREATE_ENTAILMENT 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_entailment('rdfstst_idx', sem_models('my_model'), sem_
rulebases('RDFS'));
```

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_entailment('rdfstst_idx', sem_models('my_model'), sem_
rulebases('RDFS'), SEM_APIS.REACH_CLOSURE, -
 'RDFS4A-, RDFS4B-, RDFS6-, RDFS8-, RDFS10-, RDFS13-');
```

2.2.7 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 Chapter 4) 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.

2.2.8 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_ENTAILMENT 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 2–6 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_ENTAILMENT procedure.

Example 2–6 Performing Selective Inferencing

```
EXECUTE sem_apis.create_rulebase('my_rulebase');

EXECUTE sem_apis.create_entailment('owltst_idx', sem_models('owltst'), sem_
rulebases('my_rulebase'), SEM_APIS.REACH_CLOSURE, 'SCOH,SPOH');
```

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, 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_entailment('owltst_idx', sem_models('owltst'), sem_
rulebases('OWLPRIME'), SEM_APIS.REACH_CLOSURE, 'SCOH-');
```

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_ENTAILMENT procedure.

2.3 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.

2.3.1 Using the SEM_RELATED Operator

Referring to the cancer ontology example in Section 2.1.1, 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 Section 2.1.1) 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';
```

However, many rows in the patient data table are relevant, because their diagnoses fall under this class. Example 2–7 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 Section 2.3.5.)

Example 2–7 SEM_RELATED Operator

```
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 rules index (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 rules index

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 rules index 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 Section 2.3.2 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`.

2.3.2 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 2–8). 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 2–8 expands Example 2–7 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 cancer ontology described in Section 2.1.1, the distance between `AIDS` and `Immune_System_Disorder` is 3.

Example 2–8 SEM_DISTANCE Ancillary Operator

```
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
  WHERE SEM_DISTANCE(123) <= 3;
```

Example 2–9 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.

Example 2–9 Using SEM_DISTANCE to Restrict the Number of Rows Returned

```
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 2–9, 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 2–8.

2.3.2.1 Computation of Distance Information

Distances are generated for the following properties during inference (entailment): 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.

2.3.3 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 2–10 creates a semantic index named DIAGNOSIS_SEM_IDX on the DIAGNOSIS column of the PATIENTS table using the `Cancer_Ontology` ontology.

Example 2–10 Creating a Semantic Index

```
CREATE INDEX diagnosis_sem_idx
  ON patients (diagnosis)
  INDEXTYPE IS MDSYS.SEM_INDEXTYPE;
```

The column on which the index is built (DIAGNOSIS in Example 2–10) 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.

To improve the performance of certain semantic queries, you can cause statistical information to be generated for the semantic index by specifying one or more models and rulebases when you create the index. Example 2–11 creates an index that will also generate statistics information for the specified model and rulebase. The index can be used with other models and rulebases during query, but the statistical information will be used only if the model and rulebase specified during the creation of the index are the same model and rulebase specified in the query.

Example 2–11 Creating a Semantic Index Specifying a Model and Rulebase

```
CREATE INDEX diagnosis_sem_idx
  ON patients (diagnosis)
  INDEXTYPE IS MDSYS.SEM_INDEXTYPE('ONTOLOGY_MODEL(medical_ontology),
    RULEBASE(OWLPrime)');
```

The statistical information is useful for queries that return top-k results sorted by semantic distance. Example 2–12 shows such a query.

Example 2–12 Query Benefitting from Generation of Statistical Information

```
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);
```

2.3.4 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.

2.3.5 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 2–13 creates a semantic index that uses a URI prefix.

Example 2–13 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/>)');
```

Note that 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.

3 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 Chapter 1, "Oracle Semantic Technologies Overview" and Chapter 2, "OWL Concepts".

This chapter provides reference information about the subprograms, listed in alphabetical order.

SEM_APIS.ADD_SEM_INDEX

Format

```
SEM_APIS.ADD_SEM_INDEX(
    index_code     IN VARCHAR2);
```

Description

Creates creates a semantic network index that results in creation of a nonunique B-tree index in UNUSABLE status for each of the existing models and rules indexes of the semantic network.

Parameters

index_code
Index code string.

Usage Notes

You must have DBA privileges to call this procedure.

For an explanation of semantic network indexes, see Section 1.8.

Examples

The following example creates a semantic network index with the index code string `pcsm` on the models and rules indexes of the semantic network.

```
EXECUTE SEM_APIS.ADD_SEM_INDEX('pscm');
```

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);
```

Description

Alters a semantic network index on a model.

Parameters

model_name
Name of the model.

index_code
Index code string.

command
A command that is valid for the Oracle SQL ALTER INDEX statement (for example, REBUILD or UNUSABLE). For information about the ALTER INDEX statement, see *Oracle Database SQL Language Reference*.

Usage Notes

For an explanation of semantic network indexes, see Section 1.8.

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');
```

SEM_APIS.ALTER_SEM_INDEX_ON_RULES_INDEX

Format

```
SEM_APIS.ALTER_SEM_INDEX_ON_RULES_INDEX(
    rules_index_name  IN VARCHAR2,
    index_code        IN VARCHAR2,
    command           IN VARCHAR2);
```

Description

Alters a semantic network index on a rules index.

Parameters

rules_index_name
Name of the rules index.

index_code
Index code string.

command
A command that is valid for the Oracle SQL ALTER INDEX statement (for example, REBUILD or UNUSABLE). For information about the ALTER INDEX statement, see *Oracle Database SQL Language Reference*.

Usage Notes

For an explanation of semantic network indexes, see Section 1.8.

Examples

The following example rebuilds (and makes usable if it is unusable) the semantic network index on the rules index named `rdfs_rix_family`.

```
EXECUTE SEM_APIS.ALTER_SEM_INDEX_ON_RULES_INDEX('rdfs_rix_family', 'pscm',
'rebuild');
```

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);
```

Description

Collects 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.

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 model. The DBMS_STATS.GATHER_TABLE_STATS procedure is documented in *Oracle Database PL/SQL Packages and Types Reference*.

Examples

The following example collects statistics for the semantic model named `family`.

```
EXECUTE SEM_APIS.ANALYZE_MODEL('family');
```

SEM_APIS.ANALYZE_RULES_INDEX

Format

```
SEM_APIS.ANALYZE_RULES_INDEX(
    index_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);
```

Description

Collects statistics for a specified rules index (entailment).

Parameters

index_name
Name of the rules index.

estimate_percent
Percentage of rows to estimate in the internal table partition containing information about the rules index (`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 rules index:

- `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 rules index. 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 rules index. 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 rules index is locked; `FALSE` (the default) does not gather statistics if the rules index is locked.

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 rules index. The DBMS_STATS.GATHER_TABLE_STATS procedure is documented in *Oracle Database PL/SQL Packages and Types Reference*.

For information about rules indexes, see Section 1.3.7.

Examples

The following example collects statistics for the rules index named `rdfs_rix_family`.

```
EXECUTE SEM_APIS.ANALYZE_RULES_INDEX('rdfs_rix_family');
```

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 BINARY_INTEGER DEFAULT NULL);
```

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:

- `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.
- `<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)

Usage Notes

You must first load semantic data into a staging table before calling this procedure. See Section 1.7.1 for more information.

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');
```

SEM_APIS.CLEANUP_FAILED

Format

```
SEM_APIS.CLEANUP_FAILED(
   rdf_object_type  IN VARCHAR2,
   rdf_object_name  IN VARCHAR2);
```

Description

Drops (deletes) a specified rulebase or rules index if it is in a failed state.

Parameters

rdf_object_type
Type of the RDF object: `RULEBASE` for a rulebase or `RULES_INDEX` for a rules index.

rdf_object_name
Name of the RDF object of type `rdf_object_type`.

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 rules index is in a failed state if a system failure occurred during the creation of that object. You can check if a rulebase or rules index 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 Section 1.3.6) or the SDO_RULES_INDEX_INFO view (described in Section 1.3.7), respectively.

If the rulebase or rules index 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.

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');
```

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);
```

Description

Creates a rules index (entailment) that can be used to perform OWL or RDFS inferencing, and optionally use user-defined rules.

Parameters

index_name_in
Name of the rules index (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 Section 1.3.6.

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 rules index 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. Is 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 `PROOF`, `DISTANCE`, `ENTAIL_ANYWAY`, and `USER_RULES`. See the Usage Notes for explanations of each value.

Usage Notes

For the `inf_components_in` parameter, you can specify any combination of the following keywords: `SCOH, COMPH, DISJH, SYMMH, INVH, SPIH, SPOH, DOMH, RANH, EQCH, EQPH, FPH, IFPH, SAMH, DOM, RAN, SCO, DISJ, INV, SPO, FP, IFP, SYMM, TRANS, DIF, SAM, RDFS2, RDFS3, RDFS4a, RDFS4b, RDFS5, RDFS6, RDFS7, RDFS8, RDFS9, RDFS10, RDFS11, RDFS12, RDFS13, RDFP1, RDFP2, RDFP3, RDFP4, RDFP6, RDFP7, RDFP8AX, RDFP8BX, RDFP9, RDFP10, RDFP11, RDFP12A, RDFP12B, RDFP12C, RDFP13A, RDFP13B, RDFP13C, RDFP14A, RDFP14BX, RDFP15, RDFP16`. For an explanation of the meaning of these keywords, see Table 3–1, 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`. Note that component `SAM` is not in this default list, because it tends to generate many new triples for some ontologies.

Table 3–1 Inferencing Keywords for inf_components_in Parameter

Keyword	Explanation
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.
IFP	Performs instance-level inference using instances of owl:InverseFunctionalProperty.
IFPH	Performs inference using instances of owl:InverseFunctionalProperty.
INV	Performs instance-level inference using owl:inverseOf assertions.
INVH	Performs inference based on owl:inverseOf assertions and their interactions with other language constructs.
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 3–1 (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.
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.
SYMM	Performs instance-level inference using instances of owl:SymmetricProperty.
SYMH	Performs inference for properties of type owl:SymmetricProperty.
TRANS	Calculates transitive closure for instances of owl:TransitiveProperty.

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:

- `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.
- `DISTANCE=T` generates ancillary distance information that is useful for semantic operators.
- `ENTAIL_ANYWAY=T` forces OWL inferencing to proceed and reuse existing inferred data (rules index) when the rules index has a valid status. By default, SEM_APIS.CREATE_ENTAILMENT quits immediately if there is already a valid rules index for the combination of models and rulebases.
- `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.

Examples

The following example creates a rules index 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');
```

SEM_APIS.CREATE_RULEBASE

Format

```
SEM_APIS.CREATE_RULEBASE(
    rulebase_name  IN VARCHAR2);
```

Description

Creates a rulebase.

Parameters

rulebase_name
Name of the rulebase.

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 Section 1.3.6. The SEM_MATCH table function is described in Section 1.6,

Examples

The following example creates a rulebase named family_rb. (It is an excerpt from Example 1–14 in Section 1.10.2.)

```
EXECUTE SEM_APIS.CREATE_RULEBASE('family_rb');
```

SEM_APIS.CREATE_RULES_INDEX

Format

```
SEM_APIS.CREATE_RULES_INDEX(
    index_name_in  IN VARCHAR2,
    models_in  IN SEM_MODELS,
    rulebases_in  IN SEM_RULEBASES);
```

Description

Creates a rules index based on data in one or more models and one or more rulebases.

Parameters

index_name_in
Name of the rules index.

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 Section 1.3.6.

Usage Notes

This procedure creates a rules index. For information about rules indexes, see Section 1.3.7.

Examples

The following example creates a rules index named `family_rb_rix_family`, using the `family` model and the `RDFS` and `family_rb` rulebases. (This example is an excerpt from Example 1–14 in Section 1.10.2.)

```
BEGIN
  SEM_APIS.CREATE_RULES_INDEX(
    'rdfs_rix_family',
    SEM_Models('family'),
    SEM_Rulebases('RDFS','family_rb'));
END;
/
```

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);
```

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.

column_name
Name of the column of type SDO_RDF_TRIPLE_S in `table_name`.

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.

Usage Notes

You must create the table to hold references to semantic technology data before calling this procedure to create the semantic technology model. For more information, see Section 1.9.

This procedure adds the model to the MDSYS.SEM_MODEL$ view, which is described in Section 1.3.1.

This procedure is the only supported way to create a model. Do not use SQL INSERT statements with the MDSYS.SEM_MODEL$ view.

To delete a model, use the SEM_APIS.DROP_SEM_MODEL procedure.

Examples

The following example creates a semantic technology model named `articles`. References to the triple data for the model will be stored in the TRIPLE column of the ARTICLES_RDF_DATA table. (This example is an excerpt from Example 1–13 in Section 1.10.2.)

```
EXECUTE SEM_APIS.CREATE_SEM_MODEL('articles', 'articles_rdf_data', 'triple');
```

The definition of the ARTICLES_RDF_DATA table is as follows:

```
CREATE TABLE articles_rdf_data (id NUMBER, triple SDO_RDF_TRIPLE_S);
```

SEM_APIS.CREATE_SEM_NETWORK

Format

SEM_APIS.CREATE_SEM_NETWORK(

tablespace_name IN VARCHAR2

Description

Adds semantic technology support to the database.

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.

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 in order to call this procedure, and you should call the procedure only once for the database.

To remove semantic technology support from the database, you must connect as a user with DBA privileges and call the SEM_APIS.DROP_SEM_NETWORK procedure.

Examples

The following example creates a tablespace for semantic technology system tables and adds semantic technology support to the database.

```
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');
```

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);
```

Description

Creates a virtual model containing the specified semantic models and rulebases.

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)`. At least one semantic model must be specified.

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 Section 1.3.6.

options
(Reserved for future use.)

Usage Notes

For an explanation of virtual models, including usage information, see Section 1.3.8.

A rules index 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 rules indexes, or (B) a user with DBA privileges.

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 rules index. This view may contain duplicates.
- SEMU_*vm_name*, which corresponds to a UNION of the triples in each model and rules index. This view will not contain duplicates (thus, the *U* in SEMU indicates *unique*).

  However, the SEMU_*vm_name* view is not created if the virtual model contains only one semantic model and no rules index.

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.

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'));
```

SEM_APIS.DROP_ENTAILMENT

Format

```
SEM_APIS.DROP_ENTAILMENT(
   index_name_in      IN VARCHAR2);
```

Description

Drops (deletes) a rules index (entailment).

Parameters

index_name_in
Name of the rules index (entailment) to be deleted.

Usage Notes

You can use this procedure to delete an entailment that you created using the SEM_APIS.CREATE_ENTAILMENT procedure.

Examples

The following example deletes a rules index named `OWLTST_IDX`.

```
EXECUTE sem_apis.drop_entailment('owltst_idx');
```

SEM_APIS.DROP_RULEBASE

Format

```
SEM_APIS.DROP_RULEBASE(
   rulebase_name  IN VARCHAR2);
```

Description

Deletes a rulebase.

Parameters

rulebase_name
Name of the rulebase.

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 Section 1.3.6.

Only the creator of a rulebase can delete the rulebase.

Examples

The following example drops the rulebase named `family_rb`.

```
EXECUTE SEM_APIS.DROP_RULEBASE('family_rb');
```

SEM_APIS.DROP_RULES_INDEX

Format

```
SEM_APIS.DROP_RULES_INDEX(
   index_name  IN VARCHAR2);
```

Description

Deletes a rules index.

Parameters

index_name
Name of the rules index.

Usage Notes

This procedure deletes the specified rules index, making it no longer available for use with queries against RDF data. For information about rules indexes, see Section 1.3.7.

Only the owner of a rulebase can call this procedure to drop the rules index. However, a rules index can be dropped implicitly if an authorized user drops any model or rulebase on which the rules index is based; in such a case, the rules index is dropped automatically.

Examples

The following example drops the rules index named `rdfs_rix_family`.

```
EXECUTE SEM_APIS.DROP_RULES_INDEX ('rdfs_rix_family');
```

SEM_APIS.DROP_SEM_INDEX

Format

```
SEM_APIS.DROP_SEM_INDEX(
     index_code     IN VARCHAR2);
```

Description

Drops a semantic network index on the models and rules indexes 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.

Usage Notes

For an explanation of semantic network indexes, see Section 1.8.

Examples

The following example drops a semantic network index with the index code string `pcsm` on the models and rules indexes of the semantic network.

```
EXECUTE SEM_APIS.DROP_SEM_INDEX('pscm');
```

SEM_APIS.DROP_SEM_MODEL

Format

```
SEM_APIS.DROP_SEM_MODEL(
   model_name    IN VARCHAR2);
```

Description

Drops (deletes) a semantic technology model.

Parameters

model_name
Name of the model.

Usage Notes

This procedure deletes the model from the MDSYS.SEM_MODEL$ view, which is described in Section 1.3.1.

This procedure is the only supported way to delete a model. Do not use SQL DELETE statements with the MDSYS.SEM_MODEL$ view.

Only the creator of a model can delete the model.

Examples

The following example drops the semantic technology model named `articles`.

```
EXECUTE SEM_APIS.DROP_SEM_MODEL('articles');
```

SEM_APIS.DROP_SEM_NETWORK

Format

SEM_APIS.DROP_SEM_NETWORK();

Description

Removes semantic technology support from the database.

Parameters

None.

Usage Notes

To remove semantic technology support from the database, you must connect as a user with DBA privileges and call this procedure.

Before you call this procedure, be sure to delete all semantic technology models and rulebases.

Examples

The following example removes semantic technology support from the database.

```
EXECUTE SEM_APIS.DROP_SEM_NETWORK;
```

SEM_APIS.DROP_USER_INFERENCE_OBJS

Format

```
SEM_APIS.DROP_USER_INFERENCE_OBJS(
   uname  IN VARCHAR2);
```

Description

Drops (deletes) all rulebases and rules index 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.)

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 rules indexes owned by that user.

Examples

The following example deletes all rulebases and rules indexes owned by user `SCOTT`.

```
EXECUTE SEM_APIS.DROP_USER_INFERENCE_OBJS('SCOTT');

PL/SQL procedure successfully completed.
```

SEM_APIS.DROP_VIRTUAL_MODEL

Format

```
SEM_APIS.DROP_VIRTUAL_MODEL(
   vm_name  IN VARCHAR2);
```

Description

Drops (deletes) a virtual model.

Parameters

vm_name
Name of the virtual model to be deleted.

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 rules index 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 Section 1.3.8.

Examples

The following example deletes a virtual model named `VM1`.

```
EXECUTE sem_apis.drop_virtual_model('vm1');
```

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 MDSYS.SEM_MODEL$ view, which is described in Section 1.3.1.

Examples

The following example returns the model ID number for the model named `articles`. (This example is an excerpt from Example 1–13 in Section 1.10.2.)

```
SELECT SEM_APIS.GET_MODEL_ID('articles') AS model_id FROM DUAL;

  MODEL_ID
----------
         1
```

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 MDSYS.SEM_MODEL$ view, which is described in Section 1.3.1.

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 1–13 in Section 1.10.2.)

```
SQL> SELECT SEM_APIS.GET_MODEL_NAME(1) AS model_name FROM DUAL;

MODEL_NAME
--------------------------------------------------------------------------------
ARTICLES
```

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 MDSYS.SEM_MODEL$ view, which is described in Section 1.3.1.

model_name
Name of the semantic technology model. Must match a value in the MODEL_NAME column of the MDSYS.SEM_MODEL$ view, which is described in Section 1.3.1.

subject
Subject. Must match a value in the VALUE_NAME column of the MDSYS.RDF_VALUE$ table, which is described in Section 1.3.2.

property
Property. Must match a value in the VALUE_NAME column of the MDSYS.RDF_VALUE$ table, which is described in Section 1.3.2.

object
Object. Must match a value in the VALUE_NAME column of the MDSYS.RDF_VALUE$ table, which is described in Section 1.3.2.

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 1–13 in Section 1.10.2.)

```
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
```

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 MDSYS.SEM_MODEL$ view, which is described in Section 1.3.1.

model_name
Name of the semantic technology model. Must match a value in the MODEL_NAME column of the MDSYS.SEM_MODEL$ view, which is described in Section 1.3.1.

subject
Subject. Must match a value in the VALUE_NAME column of the MDSYS.RDF_VALUE$ table, which is described in Section 1.3.2.

property
Property. Must match a value in the VALUE_NAME column of the MDSYS.RDF_VALUE$ table, which is described in Section 1.3.2.

object
Object. Must match a value in the VALUE_NAME column of the MDSYS.RDF_VALUE$ table, which is described in Section 1.3.2.

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 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 1–13 in Section 1.10.2.)

```
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)
```

SEM_APIS.LOOKUP_RULES_INDEX

Format

SEM_APIS.LOOKUP_RULES_INDEX (

models IN SEM_MODELS,

rulebases IN SEM_RULEBASES

) RETURN VARCHAR2;

Description

Returns the name of the 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 Section 1.3.6.

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 rules index that is based on the `family` model and the `RDFS` and `family_rb` rulebases. (It is an excerpt from Example 1–14 in Section 1.10.2.)

```
SELECT SEM_APIS.LOOKUP_RULES_INDEX(SEM_MODELS('family'),
  SEM_RULEBASES('RDFS','family_rb')) AS lookup_rules_index FROM DUAL;

LOOKUP_RULES_INDEX
--------------------------------------------------------------------------------
RDFS_RIX_FAMILY
```

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
    ) RETURN RDF_LONGVARCHARARRAY;
```

Description

Validates rules indexes (entailment) 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 Section 1.3.6.

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 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 entailment. For more information, see Section 2.2.5.

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.

Examples

For an example of this procedure, see Example 2–5 in Section 2.2.5.

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;
```

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 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 Section 2.2.5.

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;
```

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 MDSYS.RDF_VALUE$ table.

Parameters

value_name
Value name. Must match a value in the VALUE_NAME column in the MDSYS.RDF_VALUE$ table, which is described in Section 1.3.2.

value_type
Value type. Must match a value in the VALUE_TYPE column in the MDSYS.RDF_VALUE$ table, which is described in Section 1.3.2.

Usage Notes

This function usually causes an index on the MDSYS.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 MDSYS.RDF_VALUE$ table with a prefix value same as that returned by the VALUE_NAME_PREFIX function. This query uses an index on the MDSYS.RDF_VALUE$ table, thereby providing efficient lookup.

```
SELECT value_name FROM MDSYS.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.
```

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 MDSYS.RDF_VALUE$ table.

Parameters

value_name
Value name. Must match a value in the VALUE_NAME column in the MDSYS.RDF_VALUE$ table, which is described in Section 1.3.2.

value_type
Value type. Must match a value in the VALUE_TYPE column in the MDSYS.RDF_VALUE$ table, which is described in Section 1.3.2.

Usage Notes

This function usually causes an index on the MDSYS.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 MDSYS.RDF_VALUE$ table with a suffix value same as that returned by the VALUE_NAME_SUFFIX function. This query uses an index on the MDSYS.RDF_VALUE$ table, thereby providing efficient lookup.

```
SELECT value_name FROM MDSYS.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
```

4 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 Chapter 1, "Oracle Semantic Technologies Overview" and Chapter 2, "OWL Concepts".

This chapter provides reference information about the subprograms, listed in alphabetical order.

SEM_PERF.GATHER_STATS

Format

SEM_PERF.GATHER_STATS();

Description

Gathers statistics about RDF and OWL tables and their indexes.

Parameters

None.

Usage Notes

To use this procedure, you must connect as a user with permission to execute it. By default, when Spatial is installed as part of Oracle Database, only the MDSYS user can execute this procedure; however execution permission on this procedure can be granted to users as needed.

This procedure collects statistical information that can help you to improve inferencing performance, as explained in Section 2.2.7. 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.

Examples

The following example gathers statistics about RDF and OWL related tables and their indexes.

```
EXECUTE SEM_PERF.GATHER_STATS;
```

Index

A

ADD_SEM_INDEX procedure, 3-2
aliases
  SEM_ALIASES and SEM_ALIAS data types, 1-18
ALLOW_DUP=Y
  query option for SEM_MATCH, 1-18
ALTER_SEM_INDEX_ON_MODEL procedure, 3-3
ALTER_SEM_INDEX_ON_RULES_INDEX procedure, 3-4
ANALYZE_MODEL procedure, 3-5
ANALYZE_RULES_INDEX procedure, 3-7

B

batch (bulk) loading, 3-9
batch loading semantic data, 1-23
blank nodes, 1-7
  constructor for reusing, 1-16
bulk loading, 3-9
bulk loading semantic data, 1-22
BULK_LOAD_FROM_STAGING_TABLE procedure, 3-9

C

canonical forms, 1-6
catsem10i.sql script, 1-36
catsem11i.sql script, 1-36
CLEANUP_FAILED procedure, 3-11
constructors for semantic data, 1-14
CREATE_ENTAILMENT procedure, 3-12
CREATE_RULEBASE procedure, 3-15
CREATE_RULES_INDEX procedure, 3-16
CREATE_SEM_MODEL procedure, 3-17
CREATE_SEM_NETWORK procedure, 3-18
CREATE_VIRTUAL_MODEL procedure, 3-19

D

data types
  for literals, 1-6
data types for semantic data, 1-14
demo files
  semantic data, 1-27
DROP_ENTAILMENT procedure, 3-21
DROP_RULEBASE procedure, 3-22
DROP_RULES_INDEX procedure, 3-23
DROP_SEM_INDEX procedure, 3-24
DROP_SEM_MODEL procedure, 3-25
DROP_SEM_NETWORK procedure, 3-26
DROP_USER_INFERENCE_OBJS procedure, 3-27
DROP_VIRTUAL_MODEL procedure, 3-28
duplicate triples
  checking for, 1-6

E

entailment rules, 1-8
examples
  PL/SQL, 1-27
exporting semantic data, 1-22

F

failed state
  rulebase or rules index, 3-11
filter
  attribute of SEM_MATCH, 1-18

G

GATHER_STATS procedure, 4-2
GET_MODEL_ID function, 3-29
GET_MODEL_NAME function, 3-30
GET_TRIPLE_ID function, 3-31

H

HINT0
  query option for SEM_MATCH, 1-18

I

index_status
  attribute of SEM_MATCH, 1-18, 2-10
inferencing, 1-7
inverseOf keyword
  using to force use of semantic index, 2-13
IS_TRIPLE function, 3-33

L

literals

data types for, 1-6
loading semantic data, 1-22
bulk, 3-9
LOOKUP_RULES_INDEX procedure, 3-35

M

metadata
semantic, 1-3
metadata tables and views for semantic data, 1-14
methods for semantic data, 1-14
model ID
getting, 3-29
model name
getting, 3-30
models, 1-3
creating, 3-17
deleting (dropping), 3-25
disabling support in the database, 3-26
enabling support in the database, 3-18
SEM_MODELS data type, 1-17
SEMI_rules-index-name view, 1-10
SEMM_model-name view, 1-4
virtual, 1-11

O

objects, 1-7
options
attribute of SEM_MATCH, 1-18
Oracle Spatial
prerequisite software for RDF and OWL capabilities, 0-ix, 1-2
OWL
queries using the SEM_DISTANCE ancillary operator, 2-11
queries using the SEM_RELATED operator, 2-9

P

Partitioning
must be enabled for RDF and OWL, 0-ix, 1-2
properties, 1-7

Q

queries
using the SEM_DISTANCE ancillary operator, 2-11
using the SEM_MATCH table function, 1-17
using the SEM_RELATED operator, 2-9

R

RDF rulebase
subset of RDFS rulebase, 1-8
RDF$ET_TAB table, 1-23
RDF_VALUE$ table, 1-5
RDFS entailment rules, 1-8
RDFS rulebase
implements RDFS entailment rules, 1-8
rulebases, 1-7
attribute of SEM_MATCH, 2-10
deleting if in failed state, 3-11
SEM_RULEBASE_INFO view, 1-9
SEM_RULEBASES data type, 1-17
SEMR_rulebase-name view, 1-9
rules, 1-7
rules indexes, 1-10
deleting if in failed state, 3-11
incomplete status, 1-18, 2-10
invalid status, 1-18, 2-10
SEM_RULES_INDEX_DATASETS view, 1-11
SEM_RULES_INDEX_INFO view, 1-10

S

security considerations, 1-13
SEM_ALIAS data type, 1-18
SEM_ALIASES data type, 1-18
SEM_APIS package
ADD_SEM_INDEX, 3-2
ALTER_SEM_INDEX_ON_MODEL, 3-3
ALTER_SEM_INDEX_ON_RULES_INDEX, 3-4
ANALYZE_MODEL, 3-5
ANALYZE_RULES_INDEX, 3-7
BULK_LOAD_FROM_STAGING_TABLE, 3-9
CLEANUP_FAILED, 3-11
CREATE_ENTAILMENT, 3-12
CREATE_RULEBASE, 3-15
CREATE_RULES_INDEX, 3-16
CREATE_SEM_MODEL, 3-17
CREATE_SEM_NETWORK, 3-18
CREATE_VIRTUAL_MODEL, 3-19
DROP_ENTAILMENT, 3-21
DROP_RULEBASE, 3-22
DROP_RULES_INDEX, 3-23
DROP_SEM_INDEX, 3-24
DROP_SEM_MODEL, 3-25
DROP_SEM_NETWORK, 3-26
DROP_USER_INFERENCE_OBJS, 3-27
DROP_VIRTUAL_MODEL, 3-28
GET_MODEL_ID, 3-29
GET_MODEL_NAME, 3-30
GET_TRIPLE_ID, 3-31
LOOKUP_RULES_INDEX, 3-35
reference information, 3-1, 4-1
TRIPLE, 3-33
VALIDATE_ENTAILMENT, 3-36
VALIDATE_MODEL, 3-38
VALUE_NAME_PREFIX, 3-40, 3-42
SEM_DISTANCE ancillary operator, 2-11
SEM_INDEXTYPE index type, 2-13
SEM_MATCH table function, 1-17
SEM_MODEL$ view, 1-3
virtual model entries, 1-12
SEM_MODELS data type, 1-17
SEM_PERF package
GATHER_STATS, 4-2
SEM_RELATED operator, 2-9
SEM_RULEBASE_INFO view, 1-9

SEM_RULEBASES data type, 1-17
SEM_RULES_INDEX_DATASETS view, 1-11
SEM_RULES_INDEX_INFO view, 1-10
SEM_VMODEL_DATASETS view, 1-13
SEM_VMODEL_INFO view, 1-12
semantic data
  blank nodes, 1-7
  constructors, 1-14
  data types, 1-14
  demo files, 1-27
  examples (PL/SQL), 1-27
  in the database, 1-3
  metadata, 1-3
  metadata tables and views, 1-14
  methods, 1-14
  modeling, 1-3
  objects, 1-7
  properties, 1-7
  queries using the SEM_MATCH table
    function, 1-17
  security considerations, 1-13
  statements, 1-5
  steps for using, 1-26
  subjects, 1-7
semantic index
  creating (MDSYS.SEM_INDEXTYPE), 2-13
semantic network indexes
  adding, 3-2
  altering on model, 3-3
  altering on rules index, 3-4
  dropping, 3-24
  using, 1-25
Resource Description Framework
  See semantic technologies
semantic technologies
  overview, 1-1
SEMI_rules-index-name view, 1-10
SEMM_model-name view, 1-4
SEMR_rulebase-name view, 1-9
Spatial
  prerequisite software for RDF and OWL
    capabilities, 0-ix, 1-2
staging table
  loading data from, 3-9
staging table for bulk loading semantic data, 1-22
statements
  RDF_VALUE$ table, 1-5
statistics
  gathering for RDF and OWL, 4-2
subjects, 1-7

T

triples
  constructor for inserting, 1-16
  duplication checking, 1-6
  IS_TRIPLE function, 3-33

U

URI prefix
  using when values are not stored as URIs, 2-14
URIPREFIX keyword, 2-14

V

VALIDATE_ENTAILMENT procedure, 3-36
VALIDATE_MODEL procedure, 3-38
VALUE_NAME_PREFIX function, 3-40, 3-42
virtual models, 1-11
  SEM_MODEL$ view entries, 1-12
  SEM_VMODEL_DATASETS view, 1-13
  SEM_VMODEL_INFO view, 1-12