Oracle® Spatial

Resource Description Framework (RDF)

10*g* Release 2 (10.2)

B19307-03

October 2005

Provides usage and reference information about the support for the Resource Description Framework (RDF) in the Oracle Spatial network data model.

ORACLE®

Oracle Spatial Resource Description Framework (RDF), 10*g* Release 2 (10.2)

B19307-03

Primary Author: Chuck Murray

Contributors: Nicole Alexander, Souri Das, George Eadon, Siva Ravada

Contents

List of Examples

List of Tables

List of Figures

Preface

Oracle Spatial Resource Description Framework (RDF) provides usage and reference information about the support for the Resource Description Framework (RDF) in the Oracle Spatial network data model.

Audience

This guide is intended for those who need to use the Oracle RDF object type to manage RDF data in the database.

It is helpful if you are familiar with Spatial network data model, as documented in *Oracle Spatial Topology and Network Data Models*.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. 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.

TTY Access to Oracle Support Services

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, seven days a week. For TTY support, call 800.446.2398.

Related Documents

For information about the network data model, see *Oracle Spatial Topology and Network Data Models*.

Oracle error message documentation is only available in HTML. If you only have access to the Oracle Documentation CD, you can browse the error messages by range. Once you find the specific range, use your browser's "find in page" feature to locate the specific message. When connected to the Internet, you can search for a specific error message using the error message search feature of the Oracle online documentation.

Printed documentation is available for sale in the Oracle Store at

`http://oraclestore.oracle.com/`

To download free release notes, installation documentation, white papers, or other collateral, go to the Oracle Technology Network (OTN). You must register online before using OTN; registration is free and can be done at

`http://www.oracle.com/technology/membership`

If you already have a user name and password for OTN, then you can go directly to the documentation section of the OTN Web site at

`http://www.oracle.com/technology/documentation`

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.

1 RDF Overview

This chapter describes the support for the Resource Description Framework (RDF) in the Oracle Spatial network data model. It assumes that you are familiar with the major concepts associated with RDF, such as reification, containers, and collections. This chapter does not explain these concepts in detail, but focuses instead on how the concepts are implemented in the network data model. For an excellent explanation of RDF concepts, see the World Wide Web Consortium (W3C) *RDF Primer* at `http://www.w3.org/TR/rdf-primer/`.

It is also helpful is you are familiar with the concepts and techniques of the network data model, as described in *Oracle Spatial Topology and Network Data Models*.

The PL/SQL subprograms for working with RDF are in the SDO_RDF package, which is documented in Chapter 2, and in the SDO_RDF_INFERENCE package, which is documented in Chapter 3.

This chapter contains the following major sections:

1.1 RDF and the Network Data Model

RDF is a language used to describe metadata, particularly for information found on the Web. In addition to its formal semantics, RDF has a simple data structure that is effectively modeled using a directed graph. To describe metadata in the network data model implementation of RDF, 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 logical network. In addition, Spatial maintains information about different RDF models.

RDF statements are expressed in triples: {subject or resource, predicate or property, object or value}. In this chapter {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.

In additional to logical network information, spatial information such as node locations and link geometries can be associated with the network.

1.2 RDF Data Model in the Database

There is one universe for all RDF data stored in the database. All RDF triples are parsed and stored in the system as entries in tables under the MDSYS schema. An RDF triple {subject, property, object} is treated as one database object. As a result, a single RDF document containing multiple triples results in multiple database objects.

All the subjects and objects of triples are mapped to nodes in a 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 in an RDF network are blank nodes, URIs, plain literals, and typed literals. Oracle Database has a type named URIType to hold instances of any URI (HttpUri, DBUri, and XDBUri). This type is used to store the names of the nodes and links in the RDF network.

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

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

1.2.1 Metadata for RDF Models

The MDSYS.RDF_MODEL$ system table contains information about all RDF models defined in the database. When you create a model using the SDO_RDF.CREATE_RDF_MODEL procedure, you specify a name for the model, as well as a table and column to hold references to the RDF data, and the system automatically generates a model ID.

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

The MDSYS.RDF_MODEL$ table contains the columns shown in Table 1–1.

Table 1–1 MDSYS.RDF_MODEL$ Table Columns

Column Name	Data Type	Description
OWNER	VARCHAR2(30)	Schema of the owner of the RDF model.
MODEL_ID	NUMBER	Unique model ID number, automatically generated by Spatial.
MODEL_NAME	VARCHAR2(25)	Name of the model.
TABLE_NAME	VARCHAR2(32)	Name of the table to hold references to RDF data for the model.
COLUMN_NAME	VARCHAR2(32)	Name of the column of type SDO_RDF_TRIPLE_S in the table to hold references to RDF data for the model.

When you create an RDF model, a view for the RDF 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.RDFM_*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.RDFM_model-name View Columns

Column Name	Data Type	Description
LINK_ID	NUMBER	The unique triple identifier and 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.
END_NODE_ID	NUMBER	The VALUE_ID for the text value of the object of the triple
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.
LINK_TYPE	VARCHAR2(200)	The type of predicate represented by the URI of the P_VALUE_ID. `STANDARD`, `RDF_MEMBER`, and `RDF_TYPE` are some of the supported types.
ACTIVE	VARCHAR2(1)	The status of the link in the network: contains `Y` if the link is active; contains `N` if the link is not active.
LINK_LEVEL	NUMBER	Priority level for the link; used for hierarchical modeling, so that links with higher priority levels can be considered first in computing a path.
COST	NUMBER	The number of times the triple is stored in an application table. The triple is only stored once in an RDF system table, but may exist in several rows in a user's application table.
PARENT_LINK_ID	NUMBER	Link ID of the parent link of this link
BIDIRECTED	VARCHAR2(1)	Contains `Y` if the link is bidirected (that is, can be traversed either from the start node to the end node or from the end node to the start node), or `N` if the link is unidirected (in one direction only, from the start node to the end node).
P_VALUE_ID	NUMBER	The VALUE_ID for the text value of the predicate of the triple
CONTEXT	VARCHAR2(1)	Reserved for future use; currently `D`.
REIF_LINK	VARCHAR2(1)	Reserved for future use; currently `N`.
MODEL_ID	NUMBER	The ID for the RDF graph to which the triple belongs. It logically partitions the table by RDF graphs.

1.2.2 Namespaces

The MDSYS.RDF_NAMESPACE$ system table contains information about namespaces. Namespaces are used in RDF XML documents to make the documents more readable. In the Oracle RDF data model, namespaces are stored directly with their subjects, properties, and objects. However, Oracle does not use the MDSYS.RDF_NAMESPACE$ table in any of its internal operations; the table is provided as a convenience to users who may want to store namespaces that are used in their models.

Oracle adds to the MDSYS.RDF_NAMESPACE$ table automatically when you add namespaces using the SDO_RDF.ADD_NAMESPACES procedure. Do not use SQL INSERT statements to add namespaces this table.

The MDSYS.RDF_NAMESPACE$ table contains the columns shown in Table 1–3.

Table 1–3 MDSYS.RDF_NAMESPACE$ Table Columns

Column Name	Data Type	Description
NAMESPACE_ID	NUMBER	Unique namespace ID number, automatically generated by Spatial.
NAMESPACE_ NAME	SYS.URITYPE	Name of the namespace. Currently, namespaces are used by the Spatial network data model only for cataloging.

1.2.3 Statements

The MDSYS.RDF_VALUE$ system 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 table directly. For example, do not use SQL INSERT, UPDATE, or DELETE statements with this table.

The RDF_VALUE$ table contains the columns shown in Table 1–4.

Table 1–4 MDSYS.RDF_VALUE$ Table Columns

Column Name	Data Type	Description
VALUE_ID	NUMBER	Unique value ID number, automatically generated by Spatial.
VALUE_NAME	SYS.URITYPE	Text value for one part of the triple, if the length is 4000 characters or less. (Otherwise, the text is stored in the LONG_VALUE column.)
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 characters.
LITERAL_TYPE	VARCHAR2(4000)	For typed literals, the type information; otherwise, null. For example, for a row representing a creation date of 1999-08-16, the VALUE_TYPE column can contain `TL`, and the LITERAL_TYPE column can contain `http://www.w3.org/2001/XMLSchema#date`.
LANGUAGE_ TYPE	VARCHAR2(80)	Language tag (for example, `fr` for French) for a literal with a language tag (that is, if VALUE_TYPE is `PL@` or `PLL@`). Otherwise, this column has a null value.
LONG_VALUE	CLOB	The character string if the length of the part of the triple is greater than 4000 characters. Otherwise, this column has a null value.

1.2.3.1 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, but no support for timezone. (Even without timezone there are still multiple representations for a single value, for example, `"2004-02-18 T15:12:54"` and `"2004-02-18 T15:12:54.0000"`.)
- DATE: xsd:date, but no support for timezone. (In this case, because timezone is not supported, all valid lexical representations are already canonical, so these can be treated like the OTHER case in the next item.)
- OTHER: Everything else. (No attempt is made to match different representations).

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.

1.2.4 Subjects and Objects

RDF subjects and objects are mapped to nodes in the network data model. 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.

1.2.5 Blank Nodes

RDF triples may have unknown subject nodes and unknown object nodes. Blank nodes are used to represent these unknown nodes. Blank nodes are also used when the relationship between a subject node and an object node is n-ary (as is the case with containers). A new entry is made for each blank node encountered in a triple.

By default, no two blank nodes corresponding to two different objects will be mapped to the same network node; however, you can optionally reuse a specific blank node. To reuse a blank node, you must use the blank node constructor (described in Section 1.4.2) to input the RDF triple. The blank node constructor is required for inserting containers (see Section 1.2.8) and collections (see Section 1.2.9).

1.2.6 Properties

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

When a triple is inserted into an RDF 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 1.2.3).

1.2.7 Reification and Reification Quads

Reification of an RDF statement is a means of providing metadata for that statement; it is the action that enables the statement to be used as the subject or object in another RDF statement.

Consider a plain English example in which the statement *PersonA is a good job candidate* is reified. Assume that the following statements are then made, using the reified statement as the object:

- *PersonB says that PersonA is a good job candidate.*
- *PersonC denies that PersonA is a good job candidate.*

For PersonB and PersonC to be able to make such statements, a **reification quad** for the statement *PersonA is a good job candidate* must exist in the database, in a form such as the following:

```
<a:Stmt1, rdf:type, rdf:Statement>
<a:Stmt1, rdf:subject, a:PersonA>
<a:Stmt1, rdf:predicate, a:CandidateQuality>
<a:Stmt1, rdf:object, "Good">
```

(In the preceding statements, `a:` represents a namespace.)

PersonB and PersonC can then make their statements about PersonA, in forms like the following:

```
<a:PersonB, a:Says, a:Stmt1>
<a:PersonC, a:Denies, a:Stmt1>
```

You can use the SDO_RDF.IS_REIFIED_QUAD function to see if a reified quad for a statement exists in the database. Using the preceding example, the following query:

```
SELECT SDO_RDF.IS_REIFIED_QUAD('candidates', 'a:PersonA', 'a:CandidateQuality',
'Good') FROM DUAL;
```

returns the following result:

```
<a:Stmt1> <rdf:type> <rdf:Statement>
```

In the current release, you cannot reify statements. However, if your data includes reified statements, the RDF model stores the reification quads.

1.2.8 Containers

A container is a resource that contains things. The contained things are called members. The members of a container may be resources (including blank nodes) or literals. RDF defines three types of containers, which are explained in the World Wide Web Consortium *RDF Primer* as follows:

- A **Bag** (a resource having type rdf:Bag) represents a group of resources or literals, possibly including duplicate members, where there is no significance in the order of the members.
- A **Sequence** or **Seq** (a resource having type rdf:Seq) represents a group of resources or literals, possibly including duplicate members, where the order of the members is significant.
- An **Alternative** or **Alt** (a resource having type rdf:Alt) represents a group of resources or literals that are alternatives (typically for a single value of a property). For example, an Alt might be used to describe alternative language translations for the title of a book, or to describe a list of alternative Internet sites at which a resource might be found. An application using a property whose value is an Alt container should be aware that it can choose any one of the members of the group as appropriate.

In the Oracle RDF data model, containers follow the general rules for triple insertion. A blank node is first created with a VALUE_NAME in the following format:

```
'_:ORABNNode_id'. Triple {_:ORABNNode_id, rdf:type, containerType}
```

Triples {`_:ORABN`*`Node_id`*`, rdf:_member#, containerValue`} are then inserted for each container. As with other triples, the containerValue object will be reused if this node already exists in the database. The links for container members have their LINK_TYPE set to `RDF_MEMBER`.

1.2.9 Collections

In the Oracle RDF data model, collections are handled similarly to containers, except an additional constraint is enforced. A blank node is first created with a VALUE_NAME in the following format:

```
'_:ORABNNode_id'. Triple {_:ORABNNode_id, rdf:type, rdf:List}
```

Triples {`_:ORABN`*`Node_id`*`, rdf:_member#, listValue`} are then inserted for each collection. As with other triples, the `listValue` object will be reused if this node already exists in the database. Because collections are closed, a constraint is enforced such that no new members can be added to the list. The links for collection members have their LINK_TYPE set to `RDF_MEMBER`.

1.2.10 Rules and Rulebases

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

- An IF side pattern for the antecedents

- 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
 SDO_RDF_Aliases (MDSYS.RDF_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. Two Oracle-supplied rulebases are provided:

- RDFS
- RDF (a subset of RDFS)

The RDFS and RDF rulebases are created when you call the SDO_RDF.CREATE_RDF_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.RDFR_RDFS and MDSYS.RDFR_RDF views.

You can also create user-defined rulebases using the SDO_RDF_INFERENCE.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.RDFR_*rulebase-name* (for example, MDSYS.RDFR_FAMILY_RB for a rulebase named `FAMILY_RB`). You must use this view to insert, delete, and modify rules in the rulebase. Each MDSYS.RDFR_*rulebase-name* view has the columns shown in Table 1–5.

Table 1–5 MDSYS.RDFR_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	SDO_RDF_ALIASES	One or more namespaces to be used. (The SDO_RDF_ALIASES data type is described in Section 1.5.)

Information about all rulebases is maintained in the MDSYS.RDF_RULEBASE_INFO view, which has the columns shown in Table 1–6 and one row for each rulebase.

Table 1–6 MDSYS.RDF_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 1–9 in Section 1.8.2.)

Example 1–1 Inserting a Rule into a Rulebase

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

INSERT INTO mdsys.rdfr_family_rb VALUES(
  'grandparent_rule',
  '(?x :parentOf ?y) (?y :parentOf ?z)',
  NULL,
  '(?x :grandParentOf ?z)',
  SDO_RDF_Aliases(SDO_RDF_Alias('','http://www.example.org/family/')));
```

You can specify one or more rulebases when calling the SDO_RDF_MATCH table function (described in Section 1.5), to control the behavior of queries against RDF 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 1–9 in Section 1.8.2.)

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(SDO_RDF_MATCH(
    '(?x :grandParentOf ?y) (?x rdf:type :Male)',
    SDO_RDF_Models('family'),
    SDO_RDF_Rulebases('RDFS','family_rb'),
    SDO_RDF_Aliases(SDO_RDF_Alias('','http://www.example.org/family/')),
    null));
```

1.2.11 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 an SDO_RDF_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 SDO_RDF_INFERENCE.CREATE_RULES_INDEX procedure. To drop (delete) a rules index, use the SDO_RDF_INFERENCE.DROP_RULES_INDEX procedure.

When you create a rules index, a view for the RDF triples associated with the rules index is also created under the MDSYS schema. This view has a name in the format RDFI_*rules-index-name*, and it is visible only to the owner of the rules index and to users with suitable privileges. Each MDSYS.RDFI_*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 RDFM_*model-name* view, which is described in Table 1–2 in Section 1.2.1.

Information about all rules indexes is maintained in the MDSYS.RDF_RULES_INDEX_INFO view, which has the columns shown in Table 1–7 and one row for each rules index.

Table 1–7 MDSYS.RDF_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
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 RDF 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.RDF_RULES_INDEX_DATASETS view. This view has the columns shown in Table 1–8 and one row for each unique combination of values of all the columns.

Table 1–8 MDSYS.RDF_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 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 1–9 in Section 1.8.2.)

Example 1–3 Creating a Rules Index

```
BEGIN
  SDO_RDF_INFERENCE.CREATE_RULES_INDEX(
```

```
    'rdfs_rix_family',
    SDO_RDF_Models('family'),
    SDO_RDF_Rulebases('RDFS','family_rb'));
END;
/
```

1.2.12 RDF Security Considerations

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

- 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 SDO_RDF_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.

1.3 RDF Metadata Tables and Views

The Spatial network data model maintains several tables and views in the MDSYS schema to hold RDF-related metadata. (Some of these tables and views are created by the SDO_RDF.CREATE_RDF_NETWORK procedure, as explained in Section 1.7, and some are created only as needed.) Table 1–9 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 1–9 RDF Metadata Tables and Views

Name	Contains Information About	Described In
RDF_MODEL$	All RDF models defined in the database	Section 1.2.1
RDFM_*model-name*	Triples in the specified model	Section 1.2.1
RDF_NAMESPACE$	Namespaces that can be used in RDF XML documents	Section 1.2.2
RDF_RULEBASE_INFO	Rulebases	Section 1.2.10
RDF_RULES_INDEX_DATASETS	Database objects used in rules indexes	Section 1.2.11

Table 1–9 (Cont.) RDF Metadata Tables and Views

Name	Contains Information About	Described In
RDF_RULES_INDEX_INFO	Rules indexes	Section 1.2.11
RDF_VALUE$	Subjects, properties, and objects used to represent RDF statements	Section 1.2.3
RDFR_*rulebase-name*	Rules in the specified rulebase	Section 1.2.10
RDFI_*rules-index-name*	Triples in the specified rules index	Section 1.2.11

1.4 RDF Data Types, Constructors, and Methods

The SDO_RDF_TRIPLE object type represents RDF data in triple format, and the SDO_RDF_TRIPLE_S object type (the _S for storage) stores persistent RDF data in the database. The SDO_RDF_TRIPLE_S type has references to the data, because the actual RDF data is stored only in the central RDF schema. This type has methods to retrieve the entire triple or part of the triple.

The SDO_RDF_TRIPLE type is used to display RDF triples, whereas the SDO_RDF_TRIPLE_S type is used to store the RDF 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_T_ID  NUMBER,  -- RDF triple ID (link ID)
  RDF_M_ID NUMBER,  -- RDF model ID
  RDF_S_ID  NUMBER,  -- Subject value ID
  RDF_P_ID NUMBER, -- Property value ID
  RDF_O_ID  NUMBER)  -- Object value ID
```

Two of the ID values stored in the SDO_RDF_TRIPLE_S type (RDF_T_ID and RDF_M_ID) together uniquely identify an RDF statement in the database.

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 1–4 shows the SDO_RDF_TRIPLE_S methods.

Example 1–4 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://www.nature.com/nature/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://www.nature.com/nature/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
```

1.4.1 Constructor for Inserting Triples

The following constructor formats are available for inserting triples into an RDF 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;
```

Example 1–5 uses the first constructor format to insert a triple.

Example 1–5 SDO_RDF_TRIPLE_S Constructor to Insert a Triple

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

1.4.2 Constructor for Reusing Blank Nodes

The following constructor formats are available for inserting triples referring to blank nodes into an RDF 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;
```

Example 1–6 uses the first constructor format to insert a triple that reuses a blank node for the subject.

Example 1–6 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));
```

1.5 Using the SDO_RDF_MATCH Table Function to Query RDF Data

To query RDF data, use the SDO_RDF_MATCH table function. This function has the following attributes:

```
SDO_RDF_MATCH(
  query  VARCHAR2,
  models  SDO_RDF_MODELS,
  rulebases  SDO_RDF_RULEBASES,
  aliases  SDO_RDF_ALIASES,
  filter  VARCHAR2,
  index_status  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 RDF model or models to use. Its data type is SDO_RDF_MODELS, which has the following definition: `TABLE OF VARCHAR2(25)`

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)`

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 SDO_RDF_ALIASES, which has the following definition: `TABLE OF RDF_ALIAS`, where each RDF_ALIAS element identifies a namespace ID and namespace value. The RDF_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 SDO_RDF_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 RDF data even when the relevant rules index 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 `INCOMPLETE` or `INVALID`. For an explanation of query behavior with different `index_status` values, see Section 1.5.1.

The SDO_RDF_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 five attributes: *var*, *var*$RDFVTYP, *var*$RDFCLOB, *var*$RDFLTYP, and *var*$RDFLANG.
- For each variable *var* that cannot take a literal value, the result elements have two attributes: *var* and *var*$RDFVTYP.

In both cases, *var* has the lexical 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.

Example 1–7 selects all grandfathers (grandparents who are male) and their grandchildren from the `family` RDF model, using inferencing from both the `RDFS` and `family_rb` rulebases. (This example is an excerpt from Example 1–9 in Section 1.8.2.)

Example 1–7 SDO_RDF_MATCH Table Function

```
SELECT x, y
  FROM TABLE(SDO_RDF_MATCH(
    '(?x :grandParentOf ?y) (?x rdf:type :Male)',
    SDO_RDF_Models('family'),
    SDO_RDF_Rulebases('RDFS','family_rb'),
    SDO_RDF_Aliases(SDO_RDF_Alias('','http://www.example.org/family/')),
    null));
```

1.5.1 Performing Queries with Incomplete or Invalid Rules Indexes

You can query RDF 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 SDO_RDF_MATCH table function. (The rules index status is stored in the STATUS column of the MDSYS.RDF_RULES_INDEX_INFO view, which is described in Section 1.2.11. The SDO_RDF_MATCH table function is described inSection 1.5.)

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 RDF 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 Loading and Exporting RDF Data

To load data RDF data into the Spatial network data model, you can use either the Java application programming interface (API), as described in Section 1.6.1, or transactional INSERT statements in SQL or PL/SQL, as described in Section 1.6.2.

To export RDF data, use the Java API, as described in Section 1.6.3.

1.6.1 Loading RDF Data

To load RDF data, the data must be in NTriple format. Two methods from the `NTripleConverter` Java class are available for loading the data:

- `nTriple2NDM(String, String, String)` loads complete statements found in the NTriple file.
- `nTriple2NDM(String, String, String, int)` loads complete statements, reusing blank nodes when the same blank node name appears in more than one incoming statement. Use this method to load data with RDF containers or collections, or to load data of unknown content.

If you use the Java API to load RDF data, the table to store references to the RDF data must have a column named ID of type NUMBER and a column named TRIPLE of type SDO_RDF_TRIPLE_S. The table can contain other columns, and the columns can be in any order, but the table must contain at least these two columns.

To convert RDF/XML data to NTriple format, you have several options, including the following:

- You can use the Jena2 `rdfparse()` class. To use this class, download Jena2 (`http://jena.sourceforge.net/downloads.html`) and follow the installation instructions. To covert an RDF/XML file to NTriple format using the `rdfparse()` class, use a command in the following format:

  ```
  java jena.rdfparse input.rdf  > output.nt
  ```

- You can edit the sample rss2insert.xsl or rssn2triple.xsl file to convert all the features in the RDF/XML file. After editing the XSLT, download an XSLT processor and follow the installation and usage instructions.

For information about using the `NTripleConverter` class, including how to compile and run an example, see the `README.txt` file in the `sdordf_converter.zip` file, which you can download from the Oracle Technology Network.

1.6.2 Loading RDF Data Using INSERT Statements

To load RDF data using INSERT statements, the data should be encoded using < > (angle brackets) for URIs, _: (underscore colon) for blank nodes, and " " (quotation marks) for literals. Spaces are not allowed in URIs or blank nodes. Use the SDO_RDF_TRIPLE_S constructor to insert the data, as described in Section 1.4.1.

> **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://www.nature.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://www.nature.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 RDF/XML data to INSERT statements, you can edit the sample `rss2insert.xsl` XSLT file to convert all the features in the RDF/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 an RDF/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
```

1.6.3 Exporting RDF Data

To output RDF data stored in the network data model to a file in NTriple 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.

1.7 Quick Start for Using RDF

To work with RDF objects in an Oracle database, follow these general steps:

1. Create a tablespace for the RDF 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 an RDF network.

   Creating an RDF network adds RDF support to an Oracle database. You must create an RDF 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 an RDF network using a tablespace named RDF_TBLSPACE (which must already exist):

   ```
   EXECUTE SDO_RDF.CREATE_RDF_NETWORK('rdf_tblspace');
   ```

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

4. Create a table to store references to the RDF 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 RDF model. It is recommended that the table include a column named ID of type NUMBER and a column named TRIPLE of type SDO_RDF_TRIPLE_S, because the default loaders provided by Oracle assume that these columns exist.

   The following example creates a table named ARTICLES_RDF_DATA:

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

5. Create an RDF model.

When you create an RDF model, you specify the model name, the table to hold references to RDF 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 SDO_RDF.CREATE_RDF_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 such queries. The following example creates such indexes:

```
-- Create indexes on the subjects, properties, objects, and
-- triple IDs in the FAMILY_RDF_DATA table.
CREATE INDEX family_sub_idx ON family_rdf_data (triple.GET_SUBJECT());
CREATE INDEX family_prop_idx ON family_rdf_data (triple.GET_PROPERTY());
CREATE INDEX family_obj_idx ON family_rdf_data (TO_CHAR(triple.GET_OBJECT()));
CREATE INDEX family_tri_idx ON family_rdf_data (triple.rdf_t_id);
```

After you create the model, you can insert triples into the table, as shown in the examples in Section 1.8.

1.8 RDF Examples

This section contains the following PL/SQL examples for the Resource Description Framework:

- Section 1.8.1, "Example: Journal Article Information"
- Section 1.8.2, "Example: Family Information"

These examples are adapted from RDF demos that are supplied with Oracle Spatial. If you installed the demo files from the Companion CD, RDF demo files are under the directory for Spatial network demo files.

1.8.1 Example: Journal Article Information

This section presents a simplified PL/SQL example of an RDF model for statements about journal articles. Example 1–8 contains descriptive comments, refer to concepts that are explained in this chapter, and uses functions and procedures documented in Chapter 2.

Example 1–8 Using an RDF Model for Journal Article Information

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

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

-- Create the RDF model.
EXECUTE SDO_RDF.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, although they are in the general
--   domain of the journal Nature (http://www.nature.com/nature/).
-- 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://www.nature.com/nature/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://www.nature.com/nature/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://www.nature.com/nature/Article1',
    'http://purl.org/dc/terms/references',
    'http://www.nature.com/nature/Article2'));

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

-- Article2 has the title "A review of ABC".
INSERT INTO articles_rdf_data VALUES (5,
  sdo_rdf_triple_s ('articles',
    'http://www.nature.com/nature/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://www.nature.com/nature/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://www.nature.com/nature/Article2',
    'http://purl.org/dc/terms/references',
    'http://www.nature.com/nature/Article3'));
```

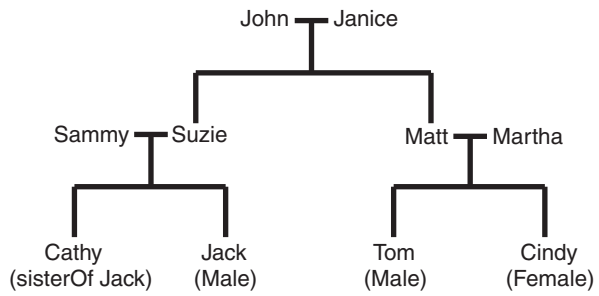
```
COMMIT;

 -- Add namespaces for cataloging.

EXECUTE SDO_RDF.ADD_NAMESPACES('http://www.w3.org-/2001/XMLSchema#', -
  'http://www.w3.org/1999/02/22-rdf-syntax-ns#', -
  'http://purl.org/dc/terms#');

-- Query RDF data.

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

SELECT SDO_RDF.GET_TRIPLE_ID(
  'articles',
  'http://www.nature.com/nature/Article2',
  'http://purl.org/dc/terms/references',
  'http://www.nature.com/nature/Article3') AS RDF_triple_id FROM DUAL;

SELECT SDO_RDF.IS_REIFIED_QUAD(
  'articles',
  'http://www.nature.com/nature/Article2',
  'http://purl.org/dc/terms/references',
  'http://www.nature.com/nature/Article3') AS is_reified_quad
FROM DUAL;

SELECT SDO_RDF.IS_TRIPLE(
  'articles',
  'http://www.nature.com/nature/Article2',
  'http://purl.org/dc/terms/references',
  'http://www.nature.com/nature/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;
```

1.8.2 Example: Family Information

This section presents a simplified PL/SQL example of an RDF model for statements about family tree (genealogy) information. Example 1–8 contains descriptive comments, refer to concepts that are explained in this chapter, and uses functions and procedures documented in Chapter 2 and Chapter 3.

The family relationships in this example reflect the family tree shown in Figure 1–1. 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 1–1 Family Tree for RDF Example



Example 1–9 Using an RDF Model for Family Information

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

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

-- Create the RDF model.
execute sdo_rdf.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
  SDO_RDF_INFERENCE.CREATE_RULES_INDEX(
    'rdfs_rix_family',
    SDO_RDF_Models('family'),
    SDO_RDF_Rulebases('RDFS'));
END;
/

-- Select all males from the family model, without inferencing.
SELECT m
  FROM TABLE(SDO_RDF_MATCH(
    '(?m rdf:type :Male)',
    SDO_RDF_Models('family'),
    null,
    SDO_RDF_Aliases(SDO_RDF_Alias('','http://www.example.org/family/')),
    null));
```

```
-- Select all males from the family model, with RDFS inferencing.
SELECT m
  FROM TABLE(SDO_RDF_MATCH(
    '(?m rdf:type :Male)',
    SDO_RDF_Models('family'),
    SDO_RDF_Rulebases('RDFS'),
    SDO_RDF_Aliases(SDO_RDF_Alias('','http://www.example.org/family/')),
    null));

-- General inferencing in the family model

EXECUTE SDO_RDF_INFERENCE.CREATE_RULEBASE('family_rb');

INSERT INTO mdsys.rdfr_family_rb VALUES(
  'grandparent_rule',
  '(?x :parentOf ?y) (?y :parentOf ?z)',
  NULL,
  '(?x :grandParentOf ?z)',
  SDO_RDF_Aliases(SDO_RDF_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 SDO_RDF_INFERENCE.DROP_RULES_INDEX ('rdfs_rix_family');

-- Re-create the rules index.
BEGIN
  SDO_RDF_INFERENCE.CREATE_RULES_INDEX(
    'rdfs_rix_family',
    SDO_RDF_Models('family'),
    SDO_RDF_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(SDO_RDF_MATCH(
    '(?x :grandParentOf ?y) (?x rdf:type :Male)',
    SDO_RDF_Models('family'),
    null,
    SDO_RDF_Aliases(SDO_RDF_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(SDO_RDF_MATCH(
    '(?x :grandParentOf ?y) (?x rdf:type :Male)',
    SDO_RDF_Models('family'),
    SDO_RDF_Rulebases('RDFS','family_rb'),
    SDO_RDF_Aliases(SDO_RDF_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.rdfr_family_rb SET
  antecedents = '(?x :parentOf ?y) (?y :parentOf ?z) (?z :height ?h)',
  filter = '(h >= 6)',
  aliases = SDO_RDF_Aliases(SDO_RDF_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 SDO_RDF_INFERENCE.DROP_RULES_INDEX ('rdfs_rix_family');

-- Re-create the rules index.
BEGIN
  SDO_RDF_INFERENCE.CREATE_RULES_INDEX(
    'rdfs_rix_family',
    SDO_RDF_Models('family'),
    SDO_RDF_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 SDO_RDF_INFERENCE.LOOKUP_RULES_INDEX(SDO_RDF_MODELS('family'),
  SDO_RDF_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(SDO_RDF_MATCH(
    '(?x :grandParentOf ?y) (?x rdf:type :Male)',
    SDO_RDF_Models('family'),
    SDO_RDF_RuleBases('RDFS','family_rb'),
    SDO_RDF_Aliases(SDO_RDF_Alias('','http://www.example.org/family/')),
    null));
```

1.9 README File for Spatial and Related Features

A `README.txt` file supplements the information in this document and the following manuals: *Oracle Spatial User's Guide and Reference, Oracle Spatial GeoRaster,* and *Oracle Spatial Topology and Network Data Models* (this manual). This file is located at:

$ORACLE_HOME/md/doc/README.txt

2

SDO_RDF Package Subprograms

The MDSYS.SDO_RDF package contains subprograms (functions and procedures) for working with the Resource Description Framework (RDF) in an Oracle database. To use the subprograms in this chapter, you must understand the conceptual and usage information in Chapter 1.

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

SDO_RDF.ADD_NAMESPACES

Format

```
SDO_RDF.ADD_NAMESPACES(
   namespace_1  IN VARCHAR2,
   namespace_2  IN VARCHAR2 DEFAULT NULL,
   namespace_3  IN VARCHAR2 DEFAULT NULL);
```

Description

Adds up to three namespaces.

Parameters

namespace_1
Namespace name (required). Must be in a valid format for a namespace.

namespace_2
Namespace name (optional). Must be in a valid format for a namespace.

namespace_3
Namespace name (optional). Must be in a valid format for a namespace.

Usage Notes

This procedure adds one, two, or three namespaces to the MDSYS.RDF_NAMESPACE$ table. (Oracle does not use the MDSYS.RDF_NAMESPACE$ table in any of its internal operations; the table is provided as a convenience to users who may want to store namespaces that are used in their models.)

There is no significance to the order in which namespaces are specified.

For information about RDF namespaces in the database, see Section 1.2.2.

Examples

The following example adds three namespaces.

```
EXECUTE SDO_RDF.ADD_NAMESPACES('http://www.w3.org-/2001/XMLSchema#', -
  'http://www.w3.org/1999/02/22-rdf-syntax-ns#', -
  'http://www.w3.org/2000/01/rdf-schema#');
```

SDO_RDF.CREATE_RDF_MODEL

Format

```
SDO_RDF.CREATE_RDF_MODEL(
   model_name   IN VARCHAR2,
   table_name   IN VARCHAR2,
   column_name  IN VARCHAR2);
```

Description

Creates an RDF model.

Parameters

model_name
Name of the model.

table_name
Name of the table to hold references to RDF data for this model.

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

Usage Notes

You must create the table to hold references to RDF data before calling this procedure to create the RDF model. For more information, see Section 1.7.

This procedure adds the model to the MDSYS.RDF_MODEL$ table, which is described in Section 1.2.1.

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

To delete a model, use the SDO_RDF.DROP_RDF_MODEL procedure.

Examples

The following example creates an RDF model named `articles`. References to the RDF 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 1–8 in Section 1.8.2.)

```
EXECUTE SDO_RDF.CREATE_RDF_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);
```

SDO_RDF.CREATE_RDF_NETWORK

Format

```
SDO_RDF.CREATE_RDF_NETWORK(
   tablespace_name  IN VARCHAR2
```

Description

Adds RDF support to the database.

Parameters

tablespace_name
Name of the tablespace to be used for tables created by this procedure.

Usage Notes

This procedure creates system tables and other database objects used for RDF support.

You should create a tablespace for the RDF 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 RDF 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 RDF support from the database, you must connect as a user with DBA privileges and call the SDO_RDF.DROP_RDF_NETWORK procedure.

Examples

The following example creates a tablespace for RDF system tables and adds RDF 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 SDO_RDF.CREATE_RDF_NETWORK('rdf_tblspace');
```

SDO_RDF.DROP_RDF_MODEL

Format

```
SDO_RDF.DROP_RDF_MODEL(
   model_name    IN VARCHAR2);
```

Description

Drops (deletes) an RDF model.

Parameters

model_name
Name of the model.

Usage Notes

This procedure deletes the model from the MDSYS.RDF_MODEL$ table, which is described in Section 1.2.1.

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

Only the creator of a model can delete the model.

Examples

The following example drops the RDF model named `articles`.

```
EXECUTE SDO_RDF.DROP_RDF_MODEL('articles');
```

SDO_RDF.DROP_RDF_NETWORK

Format

SDO_RDF.DROP_RDF_NETWORK();

Description

Removes RDF support from the database.

Parameters

None.

Usage Notes

To remove RDF 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 RDF models and rulebases.

Examples

The following example removes RDF support from the database.

```
EXECUTE SDO_RDF.DROP_RDF_NETWORK;
```

SDO_RDF.GET_MODEL_ID

Format

```
SDO_RDF.GET_MODEL_ID(
   model_name  IN VARCHAR2
   ) RETURN NUMBER;
```

Description

Returns the model ID number of an RDF model.

Parameters

model_name
Name of the RDF model.

Usage Notes

The `model_name` value must match a value in the MDSYS.RDF_MODEL$ table, which is described in Section 1.2.1.

Examples

The following example returns the model ID number for the model named `articles`. (This example is an excerpt from Example 1–8 in Section 1.8.2.)

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

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

SDO_RDF.GET_TRIPLE_ID

Format

```
SDO_RDF.GET_TRIPLE_ID(
   model_id  IN NUMBER,
   subject     IN VARCHAR2,
   property   IN VARCHAR2,
   object       IN VARCHAR2
   ) RETURN NUMBER;
```

or

```
SDO_RDF.GET_TRIPLE_ID(
   model_name  IN VARCHAR2,
   subject           IN VARCHAR2,
   property        IN VARCHAR2,
   object       IN VARCHAR2
   ) RETURN NUMBER;
```

Description

Returns the ID number of a triple in the specified RDF model, or a null value if the triple does not exist.

Parameters

model_id
ID number of the RDF model. Must match a value in the MODEL_ID column of the MDSYS.RDF_MODEL$ table, which is described in Section 1.2.1.

model_name
Name of the RDF model. Must match a value in the MODEL_NAME column of the MDSYS.RDF_MODEL$ table, which is described in Section 1.2.1.

subject
RDF subject. Must match a value in the VALUE_NAME column of the MDSYS.RDF_VALUE$ table, which is described in Section 1.2.3.

property
RDF property. Must match a value in the VALUE_NAME column of the MDSYS.RDF_VALUE$ table, which is described in Section 1.2.3.

object
RDF object. Must match a value in the VALUE_NAME column of the MDSYS.RDF_VALUE$ table, which is described in Section 1.2.3.

Usage Notes

This function has two formats, enabling you to specify the RDF 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 1–8 in Section 1.8.2.)

```
SELECT SDO_RDF.GET_TRIPLE_ID(
  'articles',
  'http://www.nature.com/nature/Article2',
  'http://purl.org/dc/terms/references',
  'http://www.nature.com/nature/Article3') AS RDF_triple_id FROM DUAL;

RDF_TRIPLE_ID
-------------
            7
```

SDO_RDF.IS_REIFIED_QUAD

Format

```
SDO_RDF.IS_REIFIED_QUAD(
   model_id  IN NUMBER,
   subject     IN VARCHAR2,
   property   IN VARCHAR2,
   object      IN VARCHAR2
   ) RETURN VARCHAR2;
```

or

```
SDO_RDF.IS_REIFIED_QUAD(
   model_name  IN VARCHAR2,
   subject          IN VARCHAR2,
   property       IN VARCHAR2,
   object      IN VARCHAR2
   ) RETURN VARCHAR2;
```

Description

Checks if all four statements that make up the reification quad for a triple in a specified model are in the database.

Parameters

model_id
ID number of the RDF model. Must match a value in the MODEL_ID column of the MDSYS.RDF_MODEL$ table, which is described in Section 1.2.1.

model_name
Name of the RDF model. Must match a value in the MODEL_NAME column of the MDSYS.RDF_MODEL$ table, which is described in Section 1.2.1.

subject
RDF subject. Must match a value in the VALUE_NAME column of the MDSYS.RDF_VALUE$ table, which is described in Section 1.2.3.

property
RDF property. Must match a value in the VALUE_NAME column of the MDSYS.RDF_VALUE$ table, which is described in Section 1.2.3.

object
RDF object. Must match a value in the VALUE_NAME column of the MDSYS.RDF_VALUE$ table, which is described in Section 1.2.3.

Usage Notes

This function returns the RDF statement if all four statements that make up the reification quad for a triple in a specified model are in the database; otherwise, it returns the string `FALSE`.

For information about reification quads, see Section 1.2.7.

Examples

The following checks if a reification quad exists for a specified statement in an RDF model named `candidates`. (This example refers to RDF statements in Section 1.2.7.)

```
SELECT SDO_RDF.IS_REIFIED_QUAD('candidates', 'a:PersonA', 'a:CandidateQuality',
'Good') FROM DUAL;
```

SDO_RDF.IS_TRIPLE

Format

```
SDO_RDF.IS_TRIPLE(
    model_id  IN NUMBER,
    subject    IN VARCHAR2,
    property   IN VARCHAR2,
    object     IN VARCHAR2) RETURN VARCHAR2;
```

or

```
SDO_RDF.IS_TRIPLE(
    model_name  IN VARCHAR2,
    subject          IN VARCHAR2,
    property        IN VARCHAR2,
    object          IN VARCHAR2) RETURN VARCHAR2;
```

Description

Checks if an RDF statement is an existing triple in the specified model in the database.

Parameters

model_id
ID number of the RDF model. Must match a value in the MODEL_ID column of the MDSYS.RDF_MODEL$ table, which is described in Section 1.2.1.

model_name
Name of the RDF model. Must match a value in the MODEL_NAME column of the MDSYS.RDF_MODEL$ table, which is described in Section 1.2.1.

subject
RDF subject. Must match a value in the VALUE_NAME column of the MDSYS.RDF_VALUE$ table, which is described in Section 1.2.3.

property
RDF property. Must match a value in the VALUE_NAME column of the MDSYS.RDF_VALUE$ table, which is described in Section 1.2.3.

object
RDF object. Must match a value in the VALUE_NAME column of the MDSYS.RDF_VALUE$ table, which is described in Section 1.2.3.

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 1–8 in Section 1.8.2.)

```
SELECT SDO_RDF.IS_TRIPLE(
  'articles',
  'http://www.nature.com/nature/Article2',
  'http://purl.org/dc/terms/references',
  'http://www.nature.com/nature/Article3') AS is_triple FROM DUAL;

IS_TRIPLE
--------------------------------------------------------------------------------
TRUE (EXACT)
```

3

SDO_RDF_INFERENCE Package Subprograms

The MDSYS.SDO_RDF_INFERENCE package contains subprograms (functions and procedures) for using the inferencing capabilities of the Resource Description Framework (RDF) in an Oracle database. To use the subprograms in this chapter, you must understand the conceptual and usage information in Chapter 1.

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

SDO_RDF_INFERENCE.CLEANUP_FAILED

Format

```
SDO_RDF_INFERENCE.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 1.2.10) or the SDO_RULES_INDEX_INFO view (described in Section 1.2.11), 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 SDO_RDF_INFERENCE.CLEANUP_FAILED('RULEBASE', 'family_rb');
```

SDO_RDF_INFERENCE.CREATE_RULEBASE

Format

```
SDO_RDF_INFERENCE.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 SDO_RDF_MATCH table function.

Rules and rulebases are explained in Section 1.2.10. The SDO_RDF_MATCH table function is described in Section 1.5,

Examples

The following example creates a rulebase named family_rb. (It is an excerpt from Example 1–9 in Section 1.8.2.)

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

SDO_RDF_INFERENCE.CREATE_RULES_INDEX

Format

```
SDO_RDF_INFERENCE.CREATE_RULES_INDEX(
   index_name_in  IN VARCHAR2,
   models_in  IN SDO_RDF_MODELS,
   rulebases_in  IN SDO_RDF_RULEBASES);
```

Description

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

Parameters

index_name_in
Name of the rules index.

models_in
One or more RDF model names. Its data type is SDO_RDF_MODELS, which has the following definition: `TABLE OF VARCHAR2(25)`

rulebases_in
One or more rulebase names. Its data type is SDO_RDF_RULEBASES, which has the following definition: `TABLE OF VARCHAR2(25)`. Rules and rulebases are explained in Section 1.2.10.

Usage Notes

This procedure creates a rules index. For information about rules indexes, see Section 1.2.11.

Examples

The following example creates a 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 1–9 in Section 1.8.2.)

```
BEGIN
  SDO_RDF_INFERENCE.CREATE_RULES_INDEX(
    'rdfs_rix_family',
    SDO_RDF_Models('family'),
    SDO_RDF_Rulebases('RDFS','family_rb'));
END;
/
```

SDO_RDF_INFERENCE.DROP_RULEBASE

Format

```
SDO_RDF_INFERENCE.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 SDO_RDF_MATCH table function. For information about rulebases, see Section 1.2.10.

Only the creator of a rulebase can delete the rulebase.

Examples

The following example drops the rulebase named `family_rb`.

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

SDO_RDF_INFERENCE.DROP_RULES_INDEX

Format

```
SDO_RDF_INFERENCE.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 1.2.11.

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 SDO_RDF_INFERENCE.DROP_RULES_INDEX ('rdfs_rix_family');
```

SDO_RDF_INFERENCE.DROP_USER_INFERENCE_OBJS

Format

```
SDO_RDF_INFERENCE.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 SDO_RDF_INFERENCE.DROP_USER_INFERENCE_OBJS('SCOTT');

PL/SQL procedure successfully completed.
```

SDO_RDF_INFERENCE.LOOKUP_RULES_INDEX

Format

```
SDO_RDF_INFERENCE.LOOKUP_RULES_INDEX (
   models      IN SDO_RDF_MODELS,
   rulebases  IN SDO_RDF_RULEBASES
   ) RETURN VARCHAR2;
```

Description

Returns the name of the rules index based on the specified models and rulebases.

Parameters

models
One or more RDF model names. Its data type is SDO_RDF_MODELS, which has the following definition: `TABLE OF VARCHAR2(25)`

rulebases
One or more rulebase names. Its data type is SDO_RDF_RULEBASES, which has the following definition: `TABLE OF VARCHAR2(25)` Rules and rulebases are explained in Section 1.2.10.

Usage Notes

For a rulebase index to be returned, it must be based on all specified models and rulebases.

Examples

The following example find the rules index that is based on the `family` model and the `RDFS` and `family_rb` rulebases. (It is an excerpt from Example 1–9 in Section 1.8.2.)

```
SELECT SDO_RDF_INFERENCE.LOOKUP_RULES_INDEX(SDO_RDF_MODELS('family'),
  SDO_RDF_RULEBASES('RDFS','family_rb')) AS lookup_rules_index FROM DUAL;

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

Index