

RDF Support in Oracle

Oracle USA Inc.

1. Introduction

Resource Description Framework (RDF) is a standard for representing information that can be identified using a Universal Resource Identifier (URI). In particular, it is intended for representing metadata about Web resources [1]. RDF is being used to represent data in numerous application areas, including Life Sciences, Digital Libraries, Geospatial Technologies, Intelligence, Electronic Commerce, and Personal Information Management. RDF provides the basic building blocks for supporting the Semantic Web [2]. The simplicity of its structure promotes interoperability across applications, and its machine-understandable format facilitates the automated processing of Web resources [3].

To represent data in RDF, each statement is broken into a <subject, predicate, object> triple. This triple is effectively modelled as a directed graph: The *subject* and *object* of the triple are modelled as nodes, and the *predicate* (or property) as a directed link that describes the relationship between the nodes. The direction of the link always points towards the object (Figure 1).

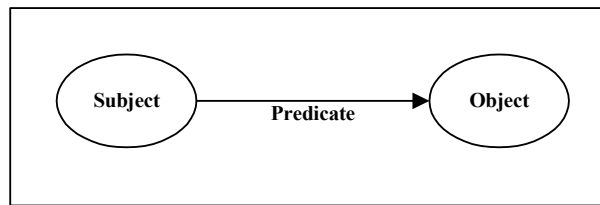


Figure 1: RDF Triple in Directed Graph Data Model

In this paper we describe Oracle's approach to managing RDF data. We introduce a new Oracle object type for storing RDF data. The RDF data store is built on top of the Oracle Spatial Network Data Model (NDM). NDM is Oracle's solution for storing, managing, and analysing networks or graphs in the database. The RDF directed graph structure is therefore effectively modelled in NDM as a directed logical network.

RDF data is stored in a central schema in Oracle: there is one universe for all RDF data in the database, and user-level access functions and constructors are provided for query and update. RDF triples are parsed and stored in the system as entries in the RDF store. Nodes in the RDF network are uniquely stored and are reused when encountered in incoming triples. A user defined application table is required to store triples corresponding to an RDF model. In these user-defined application tables, only references are stored to point to the actual triple stored in the central RDF data store. This central storage of RDF data allows analysis to be readily performed across all RDF application tables (models) in the database or on selected applications tables (models).

2. Benefits of RDF in Oracle 10g

Oracle Spatial10g release 2 introduces the industry's first open, scalable, secure and reliable data management platform for RDF-based applications. A new object types have been defined to manage RDF data in Oracle. Based on a graph data model, RDF triples are persisted, indexed and queried, similar to other object-relational data types. The Oracle10g RDF database ensures that application developers benefit from the scalability of the Oracle database to deploy scalable semantic-based enterprise applications. Moreover, applications developers benefit from a RDF data model that leverages all of the features from the Oracle10g database.

RDF Application Areas

- Web Metadata: providing information about Web resources and systems that use them (content rating, capabilities, privacy)
- Life Sciences applications: query knowledge bases that can vary from simple taxonomies to full fledged ontologies
- Social Network Applications: Friend of a Friend applications, social network tracking and navigation common in security and intelligence applications
- Semantic Information Integration: define shared, central business information model (ontology) to support information sharing across applications
- Semantic Web: Enable automated processing of Web information by software agents
- Portals and e-Marketplace Applications: ability to query large amounts of metadata

RDF Specific Functionality

- Full support for RDF statements and RDF schema
- Perform SQL search and retrieval of RDF models using graph patterns
- Supports all RDF data types (URIs, blank nodes, plain literals, typed literals, long literals, collection types)
- Incorporates inferencing of RDF schema
- Search a RDF model containing origin of data (provenance) using graph pattern
- Search RDF model by including inferred triples
- Reification of assertions (statements)
- Alias based URIs in graph pattern
- Syntax for specifying user defined rules
- Data Access: SQL and Java API
- Subsumption: Support for subClassOf and subPropertyOf
- Framework for user defined inferencing rules
- Visualization tool for representing logical graph view of RDF data

Related Oracle DBMS features

- Clustered database servers
- Partitioning: Oracle table partitioning in support of very large graphs
- Parallelism: Oracle parallelism to support load, index and query of very large graph models
- Data Loading: Import and export data in triple formats (e.g. N-triple) using Oracle's SQL Loader utility
- Spatial analysis of location-based information
- Temporal Analysis (time series, time stamping)
- Versioning
- Text Search
- Support for unstructured data types (e.g. XML, spatial, images, georaster imagery, audio, video, text)
- XML tools
- Middleware: Integrated with germane Oracle Application Server technology (BPEL, XSLT, UDDI, portal,

3. RDF Concepts

A complete description of the RDF language, including its concepts, abstract syntax, and semantics, is available on the Website: <http://www.w3.org/RDF/>. In this section we briefly review the RDF concepts (*triples*, *graphs*, *containers*, and *reification*) addressed in this paper.

Each RDF statement describing a resource is represented using a *triple*. And a set of triples is known as an RDF *graph* or *model*. The components of a triple are the *subject*, represented by a *URI* or a *blank node*; the *predicate* or *property*, represented by a *URI*; and the *object*, represented by a *URI*, a *blank node*, or a *literal*:

- A *URI* (e.g. <http://www.w3.org/1999/02/22-rdf-syntax-ns#subject>) is a more general form of Uniform Resource Locator (URL). It allows information about a resource to be recorded without the use of a specific network address.
- A *blank node* is used when a subject or object node is unknown. It is also used when the relationship between a subject and an object node is n-ary (as is the case with *RDF* containers). A blank node is prefixed with the identifier ‘_:’ and has the form *_:anyname001*.
- A *literal* is basically a string with an optional language tag. It is used to represent values like names, dates, and numbers.
- A *typed literal* (e.g. “25”^^<http://www.w3.org/2001/XMLSchema#int>) is a string combined with a datatype. The datatype is always a URI.

To describe groups of things in RDF (for example, to illustrate that a class has several students), a resource called a *container* is used. The participants of a container are called *members*. To represent a container and its members, a blank node is typically generated for the container, and each member is attached to this node as the object of a triple.

Reification is the description of a triple using the RDF vocabulary. The reification vocabulary consists of four statements, which make up the reification quad. For example, the triple $\langle S, P, O \rangle$ reified by a resource *R* is represented by the four triples:

- $\langle R, \text{rdf:type}, \text{rdf:Statement} \rangle$
- $\langle R, \text{rdf:subject}, S \rangle$
- $\langle R, \text{rdf:property}, P \rangle$
- $\langle R, \text{rdf:object}, O \rangle$

Assertions can then be made about triple $\langle S, P, O \rangle$, using *R*, for example, $\langle N, \text{ora:said}, R \rangle$. Reification therefore allows triples to be attached as properties to other triples.

4. Data Storage in Oracle

The RDF store in Oracle is built on top of the Oracle Spatial Network Data Model (NDM), in which a network or graph captures relationships between objects using connectivity. RDF graphs are modelled as a directed logical network in NDM. In this network, the subjects and objects of triples are mapped to nodes, and predicates (*or* properties) are mapped to links that have subject start-nodes and object end-nodes. A link, therefore, represents a complete RDF triple. A key feature of RDF storage in Oracle is that nodes are stored only once – regardless of the number of times they participate in triples. But a new link is always created whenever a new triple is inserted. Figure 2 illustrates how an RDF graph of three RDF triples: $\langle S1, P1, O1 \rangle$; $\langle S1, P2, O2 \rangle$; $\langle S2, P2, O2 \rangle$ is modelled. When a triple is deleted from the database, the corresponding link is removed. However, the nodes attached to this link are not removed if there are other links connected to them.

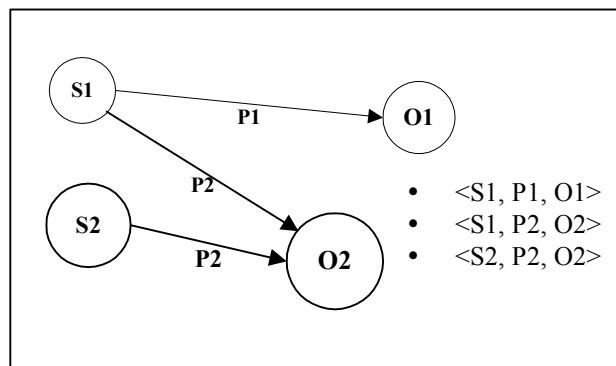


Figure 2: RDF Graph in Oracle

In Oracle, RDF triples are parsed and stored in global tables under a central schema. All the triples for all the RDF graphs (or models) in the database are stored under this schema. Only references (IDs) to these triples are stored in the user-defined application tables that may contain other attributes related to the triples. And functions are provided to retrieve the actual triples when necessary. The required tables for RDF data storage are: *RDF_MODEL\$*, *RDF_VALUE\$*, *RDF_NODE\$*, *RDF_LINK\$*, and *RDF_BLANK_NODE\$*. The *RDF_NODE\$* and *RDF_LINK\$* tables.

4.1 Inserting Triples into the RDF data store

When a user attempts to insert a triple, the `RDF_MODEL$` table is first checked to ensure that the RDF graph exists. The `RDF_VALUE$` table is then checked to determine if the text values for each part of the triple (subject, predicate, object) already exist in the database. If they already exist, their `VALUE_ID`s are retrieved. The `RDF_LINK$` table is then checked to determine if the triple already exists in the specified graph. If the triple already exists in the specified graph, the IDs for the previously inserted triple are returned and no new inserts are made into the data store.

If the text values for the triple are not found in the `RDF_VALUE$` table, they are inserted into the table and assigned new `VALUE_ID`s. The `VALUE_ID`s corresponding to the subject and the object of the triple are inserted into the `RDF_NODE$` table. In the `RDF_LINK$` table, a new `LINK_ID` is then generated for the triple. The `VALUE_ID` for the predicate of the triple becomes the `P_VALUE_ID` in the `RDF_LINK$` table, and the `VALUE_ID`s for the subject and object of the triple become the `START_NODE_ID` and `END_NODE_ID`, respectively. A new record is inserted into the `RDF_LINK$` table whenever a new triple is entered into a graph.

Inserted triples may contain blank nodes as their subject or object. When encountered, blank nodes are automatically renamed to globally unique, Oracle system-generated blank nodes of the form `_:ORABNuniqueID`. By default, blank nodes are not reused by the system. However, a user has the option to specify whether a particular blank node is to be reused by incoming triples. In such a case, the blank node's original name, along with the Oracle system-generated name and `VALUE_ID` are stored in the `RDF_BLANK_NODES$` table. Future incoming blank nodes with the same original name and belonging to the same RDF graph will cause the existing blank node in the database to be reused. To load containers, blank nodes must be reused. Entries in the `RDF_BLANK_NODES$` table can be safely deleted when their reuse is no longer required.

4.2 RDF Object Types

Two new object types have been defined to manage RDF data in Oracle: `SDO_RDF_TRIPLE` and `SDO_RDF_TRIPLE_S`. The `SDO_RDF_TRIPLE` object is defined to serve as the triple representation of RDF data <subject, predicate, object>. The `SDO_RDF_TRIPLE_S` (RDF triple storage) object is defined to store persistent RDF data in the database. The RDF triple storage object contains only IDs that point to the triple maintained in the central schema. Several constructors and member functions are provided with the storage object. The `SDO_RDF_TRIPLE` type provides a triple view of the data, and the `SDO_RDF_TRIPLE_S` types stores the IDs for the triple. The IDs stored are:

- `RDF_T_ID`: the unique triple ID, which is the `LINK_ID` in the `RDF_LINK$` table.
- `RDF_M_ID`: the `MODEL_ID` for the graph, which is the `MODEL_ID` in the `RDF_LINK$` table.
- `RDF_S_ID`: the `VALUE_ID` for the subject, which is the `START_NODE_ID` in the `RDF_LINK$` table.
- `RDF_P_ID`: the `VALUE_ID` for the predicate, which is the `P_VALUE_ID` in the `RDF_LINK$` table.
- `RDF_O_ID`: the `VALUE_ID` for the object, which is the `END_NODE_ID` in the `RDF_LINK$` table.

The member function `GET_TRIPLE()` returns the `SDO_RDF_TRIPLE` type with the subject, predicate, and object of the triple. The member functions `GET_SUBJECT()`, `GET_PROPERTY()` and `GET_OBJECT()` return the subject, predicate, and object, respectively. The `GET_OBJECT()` function returns a CLOB type, since the returned object may be a long literal.

5. Examples of Using RDF

These following steps are required to work with RDF data in an Oracle database. Since RDF data store tends to be very large, a separate tablespace for all RDF tables is recommended. Users must be connected as a user with appropriate privileges to create the tablespace. The following example creates a tablespace named `RDF_TBSPACE`:

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

1. Create an RDF network: Creating an RDF network enables RDF store in the Oracle database. Only users with DBA privilege can create an RDF network. Create the network only once for an Oracle database instance. 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');
```

If database user SCOTT will now be used to create the RDF application tables and create RDF models, user SCOTT needs this privilege. So grant this privilege while connected as a DBA user.

```
GRANT EXECUTE ON MDSYS.RDF_APIS_INTERNAL TO SCOTT;
```

2. Create a table to store references to the RDF data: Users do not need the DBA privilege for this step. 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 this table include a column named ID of type NUMBER and a column named TRIPLE of type SDO_RDF_TRIPLE_S (the default loaders provided by Oracle assume these columns). The following example creates a table named FAMILY_RDF_DATA:

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

3. Create an RDF model: An RDF model is created by specifying a model name, the table name 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 FAMILY, which will use the table created in the preceding step.

```
EXECUTE SDO_RDF.CREATE_RDF_MODEL('FAMILY', 'FAMILY_RDF_DATA', 'TRIPLE');
```

5.1 Example: Family Tree in RDF

This section presents a simple example of an RDF model for statements about a family tree (genealogy). The family relationships in this example reflect the tree shown in Figure 4. This figure also shows some of the information directly stated in the example: Cathy is the sister of Jack, both Jack and Tom are male, and Cindy is female.

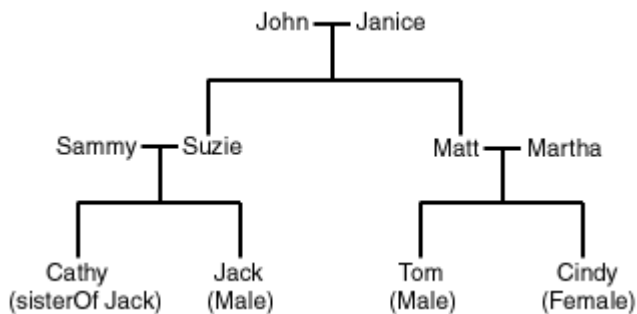


Figure 3: Family Tree

The rows inserted into the FAMILY_RDF_DATA table (shown in **Appendix A 1**) 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). After the data is inserted into the FAMILY_RDF_DATA table, inferencing queries can be executed on this data as shown in the following examples.

```
-- Select all males from the family model
SELECT m MALES
  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));

MALES
-----
http://www.example.org/family/Jack
http://www.example.org/family/Tom
```

```
-- Select all grandfathers and their grandchildren from the family model,
-- No rows will be returned as "grandParentOf" relationship is not explicitly
-- defined in the data

SELECT x grandParent, 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));

no rows selected
```

Oracle's RDF data store allows for the storage of multiple representations of literal values. For example, in the SQL insert statements in Appendix A1, the last three statements make assertions about height of Jack, Tom and Cindy. Jack and Cindy are both 6ft tall, but the values are entered differently so that lexically they look different. When this value is stored in the RDF store, a generic representation is used to treat them both as equivalent values, but at the same time, the original value is also preserved. The following query shows how to get both Jack and Cindy with a single query that asks about height. This example also shows how to use literal constants in the RDF query.

```
SELECT p Person, h Height
  FROM TABLE(SDO_RDF_MATCH(
    ' (?p :height "06.00"^^xsd:decimal) (?p :height ?h)',
    SDO_RDF_Models('family'), null,
    SDO_RDF_Aliases(SDO_RDF_Alias('', 'http://www.example.org/family/'), null));

PERSON
-----
HEIGHT
-----
http://www.example.org/family/Jack
6

http://www.example.org/family/Cindy
06.00
```

5.2 RDF Schema

RDF does not make assumptions about any particular application domain, nor does it define the semantics of any domain. For example, in this family model, there is no domain specific constraint that only males can be the objects of “fatherOf” relationship. RDF Schema lets users define the domain for an application. This information is represented using the concepts of classes, properties, and attributes.

In the family example, Person is a class that has two subclasses: Male and Female. And parentOf is a property that has two subproperties: fatherOf and motherOf. And siblingOf is a property that has two subproperties: brotherOf and sisterOf. The properties fatherOf and brotherOf have the attribute Male, and motherOf and sisterOf have the attribute Female. Appendix A2 shows triples which are inserted into the FAMILY RDF model to take advantage of the RDFS rules.

Creating Rule Indexes

RDFS inferencing rules can be used in answering RDF queries by enabling the RDFS rules for a given set of RDF models. A rule index is created for a set of RDF models and at the query time, users can optionally specify the use of RDFS rules in answering RDF queries. The rules index will pre-generate (forward chain) all the triples which can be inferred using the RDFS rules. This pre-generation process is done by creating a rules index as shown in this example:

```
-- 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;
/
```

Once this rules index is created, the inference query to find all the males from the family model can be executed with the option of using the RDFS inferencing in answering the query:

```
-- Select all males from the family model, with RDFS inferencing.
SELECT m MALES
  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));

MALES
-----
http://www.example.org/family/John
http://www.example.org/family/Matt
http://www.example.org/family/Sammy
http://www.example.org/family/Jack
http://www.example.org/family/Tom
```

In addition to the base RDFS rules, users can create their own set of rule bases and use them in inferencing queries. For example, for the family RDF model, users can define a “grandParentOf” rule to define the grand parent property. A rules index is then created to enable this rule for use in inferencing queries. The following example shows how to define a user defined rule and corresponding rules index. This time the RULES INDEX is created for RDFS and FAMILY_RB together.

```

EXECUTE SDO_RDF_INFERENCE.CREATE_RULEBASE('family_rb');

INSERT INTO mdsys.rdf_r_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/')));

EXECUTE SDO_RDF_INFERENCE.DROP_RULES_INDEX ('rdfs_rix_family');

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.
-- Use inferencing from both the RDFS and family_rb rulebases.
SELECT x grandParent, 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));

```

GRANDPARENT

GRANDCHILD

<http://www.example.org/family/John>
<http://www.example.org/family/Cindy>

<http://www.example.org/family/John>
<http://www.example.org/family/Tom>

<http://www.example.org/family/John>
<http://www.example.org/family/Cathy>

<http://www.example.org/family/John>
<http://www.example.org/family/Jack>

One can also create rule bases with a filter condition. For example, the above GRANDPARENT_RULE can be modified to find only those grand parents who are grand parents of children above 6ft tall. The following example shows the SQL for this case.

```

UPDATE mdsys.rdf_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';

-- See the result of the update.
SELECT index_name, status FROM MDSYS.RDF_RULES_INDEX_INFO;

INDEX_NAME                                STATUS
-----
RDFS_RIX_FAMILY                            INVALID

-- 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;

LOOKUP_RULES_INDEX
-----
RDFS_RIX_FAMILY

-- Select grandfathers of tall (>= 6) grandchildren, and their
-- tall grandchildren.
SELECT x grandParent, y TallGrandChild
  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));

GRANDPARENT
-----
TALLGRANDCHILD
-----
http://www.example.org/family/John
http://www.example.org/family/Cindy

http://www.example.org/family/John
http://www.example.org/family/Jack

```

Appendix A 1

```
-- 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'));

-- 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'));

```

Appendix A 2

```

-- 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'));
```

References

- [1] W3C, *RDF Primer*, <http://www.w3.org/TR/rdf-primer/>, (as of 02/24/2005).
- [2] W3C, *Semantic Web*, <http://www.w3.org/2001/sw/>, (as of 02/24/2005).
- [3] W3C, *Frequently Asked Questions about RDF*, <http://www.w3.org/RDF/FAQ2003>, (as of 02/24/2005).
- [4] Kowari, <http://www.kowari.org/>, (as of 02/24/2005).
- [5] Sesame, <http://www.openrdf.org/>, (as of 02/24/2005).
- [6] N. Alexander, X. Lopez, S. Ravada, S. Stephens, and J. Wang. *RDF Data Model in Oracle*. in *W3C Workshop on Semantic Web for Life Sciences* 2004. Cambridge, Massachusetts, USA.
- [7] Oracle Corporation, *Application Developer's Guide – Object-Relational Features*. 10g Release 1 (10.1) ed. 2003.
- [8] Oracle Corporation, *Topology and Network Data Models*. 10g Release 1 (10.1) ed. 2003.
- [9] S. Powers, *Practical RDF*. First ed. 2003: O'Reilly & Associates, Inc. 331.
- [10] J. Carroll, I. Dickinson, C. Dollin, D. Reynolds, A. Seaborne, and K. Wilkinson. *Jena: Implementing the Semantic Web Recommendations*. in *World Wide Web Conference* 2004. New York, New York, USA.
- [11] Oracle Corporation, *Oracle XML DB Developer's Guide*, 10g Release 1 (10.1) ed. 2003.