RDF Data Model in Oracle
Nicole Alexander, Xavier Lopez, Siva Ravada, Susie Stephens, Jack Wang
Oracle Corporation

1. Introduction

The Resource Description Framework (RDF) is a language for representing information about resources in the World Wide Web. To describe information in RDF, statements are essentially broken into triples: \{subject/resource, predicate/property, object/value\}. Each triple is a complete and unique fact, in a specific domain, and can be represented by a link in a directed graph. The proposed data model stores RDF triples in the Oracle database as a logical network (using Oracle Spatial Network Data Model). This document describes the proposed storage and query model for RDF in Oracle. Initially, this project will provide support for key RDF concepts: graph data model, URI-based vocabulary, datatypes, and reification. It will also support a subset of typical RDF queries in SQL.

2. RDF Data Model in the Database

RDF statements are expressed in triples: \{subject or resource, predicate or property, object or value\}. In this document \{subject, property, object\} will be used to describe a triple, and the terms statement and triple may at times 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. Oracle 10g supports a directed and undirected logical graphs (networks) as part of Oracle Spatial Network Data Model (NDM). The proposed RDF data model maps RDF triples to a logical network managed by NDM. In addition to the core data, a catalogue service is provided: by maintaining information about different RDF models, including the namespaces used in these models. RDF triple data is mapped onto a graph by storing subjects and objects as nodes, and properties as links. The storage for RDF data is managed by Oracle: all the RDF data is managed in a central schema, and user-level access functions and constructors are provided to query and update the RDF data. There is one universe for all RDF data stored in the database. Each RDF triple: \{subject, property, object\} is treated as one unique database object. As a result, a single RDF document comprising a number of triples will result in multiple database objects.

Metadata for RDF Models

A system level table is created to store information on all models defined in the database. When a new RDF model is created, an entry is made to this table. To create a model, the user specifies a name for the model and the system automatically generates a model ID. This model ID may then be used instead of the model name to refer to a particular model. Using the model ID instead of the model name will reduce space overhead.

RDF Namespaces

Namespaces are used in RDF/XML documents to make the documents readable. In Oracle’s RDF data model, namespaces are stored directly with their subjects, properties, and objects. However, an additional namespaces table is provided so that a catalog of all the namespaces used in an RDF universe can be optionally stored.

When a new namespace is inserted into the database, a namespace ID is automatically generated for the namespace. This namespace ID can then be used as a foreign key to refer to the namespace. Currently, namespaces are only used for cataloging purposes.
**RDF Statements**

RDF statements are represented in triples: subject, property, and object. In Oracle’s RDF data model, the table that stores the text values (URIs, literals, etc) for these three pieces of information is called RDF_VALUE$ and has the following columns:

```
SQL> desc RDF_VALUE$;

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>VALUE_ID</td>
<td>NUMBER</td>
</tr>
<tr>
<td>VALUE_NAME</td>
<td>URITYPE</td>
</tr>
<tr>
<td>VALUE_TYPE</td>
<td>VARCHAR2(10)</td>
</tr>
<tr>
<td>LITERAL_TYPE</td>
<td>XMLTYPE</td>
</tr>
</tbody>
</table>
```

When a new statement is inserted into an RDF model, a record is created in the RDF_VALUE$ table for each part of the triple. The text value for each part of the triple is entered into the VALUE_NAME column. Each text value is assigned a unique VALUE_ID. If a text value for a particular subject, property, or object already exists in the RDF_VALUE$ table, no new entry is made (and the existing entry is reused). The VALUE_TYPE column describes the type of text information stored in the VALUE_NAME column. Possible values for VALUE_TYPE are:

- **VALUE_TYPE=’UR’**: text value is a URI
- **VALUE_TYPE=’PL’**: text value is a plain literal
- **VALUE_TYPE=’TL’**: text value is a typed literal
- **VALUE_TYPE=’TC_BAG’**: text value is the blank node for a typed collection: rdf:Bag
- **VALUE_TYPE=’TC_ALT’**: text value is the blank node for a typed collection: rdf:Alt
- **VALUE_TYPE=’TC_SEQ’**: text value is the blank node for a typed collection: rdf:Seq
- **VALUE_TYPE=’BN’**: text value is a blank node.

If the object of a statement is a literal, it is possible that the literal is a typed literal. In this case, the type information is stored in the LITERAL_TYPE XMLType column (otherwise, the LITERAL_TYPE attribute is NULL). The RDF_VALUE$ table therefore stores all the values: **blank nodes, URIs, plain literals, typed literals and typed collections** for the nodes (subjects and objects) and links (properties) that are present in the entire RDF network.

**Typed Literals**

Typed literals are handled by storing the type information in the LITERAL_TYPE XMLType column of the RDF_VALUE$ table. Consider this example, which describes the creation date [RDF-PRIMER]:

```text
```
When an entry is made for the object of this statement, the character string ‘1999-08-16’ goes into the VALUE_NAME column and the string type ‘http://www.w3.org/2001/XMLSchema#date’ goes into the LITERAL_TYPE column (after adding <datatype> </datatype> tags to make it valid XML). The VALUE_TYPE attribute is ‘TL’.

**RDF Subjects and Objects**

Subjects and objects in Oracle’s RDF data model 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 can be both subject and object nodes. There is a NODE_ID column which is a unique ID and is the same as the VALUE_ID in the RDF_VALUE$ table. The RDF_NODE$ table stores only the VALUE_IDs that belong to nodes (i.e. subjects and objects) of statements in the database.

**Blank Nodes**

In 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 collections). In Oracle’s RDF data model, a new entry is made for each blank node encountered in an RDF triple. By default, no two blank nodes corresponding to two different objects will be mapped to the same network node. A user has the option, however, to reuse a particular blank node in a model. If the user chooses to reuse a blank node in a model, an entry is made to a blank node table. The table that stores the blank node information is called RDF_BLANK_NODE$ and has the following columns:

```
SQL> desc RDF_BLANK_NODE$;
Name                                      Type
----------------------------------------------------
NODE_ID                                   NUMBER
NODE_VALUE                                 VARCHAR2
ORIG_NAME                                  VARCHAR2
MODEL_ID                                   NUMBER
```

A blank node’s NODE_VALUE or VALUE_NAME is derived by appending the unique VALUE_ID for the node to the blank-node identifier _:blankNode. This forms a VALUE_NAME of _:blankNode\node_id. To reuse blank nodes, the blank node constructor (section 4.1) must be used to input the RDF triple. When the blank node
constructor is used, the RDF_BLANK_NODES table is searched for a blank node with the same MODEL_ID and ORIG_NAME or NODE_VALUE as the blank node being entered. If none is found, a new blank node entry is made to the RDF_VALUES table and its ID, value name and original name copied to the RDF_BLANK_NODES table to be subsequently reused when specified. Deleting blank nodes from the RDF_BLANK_NODES table will not remove the corresponding values from the RDF_VALUES and RDF_NODES table. However, it will prevent these blank nodes from being reused when the blank node constructor is utilized in the future.

**RDF Properties (or Predicates)**

RDF properties are mapped to links. When a triple is inserted into an RDF model, the subject, property, and object are first checked against the RDF_VALUES table, to see if entries for their text values already exist in the model. If they already exist (due to previous statements in other models) no new entries are made; if they do not exist, three new records are inserted into the RDF_VALUES table. If the subject, property, and object text values already exist in the RDF_VALUES table, another check is issued to determine if the actual triple exists. This second check is issued against the RDF_LINKS table. The RDF_LINKS table stores the properties and triple information, and has the following columns:

```
SQL> desc RDF_LINK$;
Name                         Type
----------------------------------
LINK_ID                       NUMBER
VALUE_ID                      NUMBER
START_NODE_ID                 NUMBER
END_NODE_ID                   NUMBER
LINK_TYPE                     VARCHAR2(10)
ACTIVE                        VARCHAR2(1)
LINK_LEVEL                    NUMBER
PARENT_LINK_ID                NUMBER
MODEL_ID                      NUMBER
```

If the triple for the particular model already exists, no new triple is inserted. Otherwise, a unique ID is generated for the new triple. This ID is stored as the LINK_ID (also known as the RDF_T_ID). The VALUE_ID in the RDF_VALUES table corresponding to the subject becomes the START_NODE_ID; and the VALUE_ID corresponding to the object becomes the END_NODE_ID for this link. The VALUE_ID is the same as the VALUE_ID in the RDF_VALUES table. The MODEL_ID column logically partitions the RDF_LINKS table. Selecting all the links for a specific MODEL_ID, returns the RDF network for that specified model.

**Reification in RDF**

A reification of a statement in RDF is a description of the statement using an RDF statement. In Oracle’s RDF data model, a hierarchical link model is used to represent statements made about other statements. The LINK_TYPE in the RDF_LINKS table can be one of two values: (i) SS, which represents a statement that is a simple statement; and (ii) RS, which represents a statement that is a reification statement. In the RS case, the link type represents the fact that the current statement makes a statement about some other statement already in the database (i.e. the current statement is not the reified statement).
To process a reification statement, the subject node, which is a URI for the creator of the reification statement, the property rdf:Statement, and a blank object node are first inserted into the RDF_VALUES table with the usual checks. An entry is then made to the RDF_LINKS table with a new LINK_ID, the VALUE_ID for rdf:Statement, subject VALUE_ID as START_NODE, and blank node VALUE_ID as END_NODE. The LINK_ID of the reified statement is then identified in the database and the PARENT_LINK_ID of the current reification statement is set to the LINK_ID of the reified statement. The reification statement is therefore the child of the reified statement. There is a many-to-one pointer in this case: the reification statement points to the reified statement. Multiple RDF statements can make assertions about other statements. This is represented in the data model as links with multiple child links. The MODEL_ID for a reification statement should match the MODEL_ID of the reified statement.

Typed Collections (RDF Containers)

There is often a need to describe groups of things: for example, to say that a book was created by several authors, or to list the students in a course or software modules in a package. RDF provides several predefined (built-in) types and properties that can be used to describe such groups. 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:

- rdf:Bag
- rdf:Seq
- rdf:Alt

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 any one of the members of the group can be chosen as appropriate [RDF-PRIMER]. Figure 1, represents the sentence “Course 6.001 has the students Amy, Mohamed, Johann, Maria, and Phuong”, using a simple bag container.
[RDF-PRIMER]

Figure 1: A Simple Bag Container Description

```xml
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:s="http://example.org/students/vocab#">
  <rdf:Description rdf:about="http://example.org/courses/6.001">
    <s:students>
      <rdf:Bag>
        <rdf:li rdf:resource="http://example.org/students/Amy"/>
        <rdf:li rdf:resource="http://example.org/students/Mohamed"/>
        <rdf:li rdf:resource="http://example.org/students/Johann"/>
        <rdf:li rdf:resource="http://example.org/students/Maria"/>
        <rdf:li rdf:resource="http://example.org/students/Phuong"/>
      </rdf:Bag>
    </s:students>
  </rdf:Description>
</rdf:RDF>
```

Example 1: RDF/XML for the Bag of Students

In Oracle’s RDF data model, these container types are handled following the general principles of triple creation, with a few extra steps: a blank node is first created with a VALUE_NAME _blankNodeNode_id, and VALUE_TYPE = TC_CollectionType. Triples {_blankNodeNode_id, rdf:blankNodeNode_id_coll#, collection_value} are then inserted for each collection member. As with other triples, the collection_value object
will be reused if this node already exists in the database. The property link for collection members has its PARENT_LINK_ID set to the LINK_ID of the originating relationship link, i.e. the link whose object (or end node) is the collection’s blank node. The collection’s blank node has a LITERAL_TYPE containing the RDF/XML for the collection (see Example 1). Example 2, shows how the rdf:Bag described in Example 1, is handled.

Example 2: Representing Collections in NDM RDF Data Model

Consistency between Links and Nodes

In Oracle’s RDF data model, a new link is always created whenever a new RDF triple is inserted into the database. However, new nodes may not always be created, since some nodes represent URIs, which are reused if they already exist in the database. RDF triples can therefore be considered unique parents of the links in a network, but the nodes may have multiple parents. When an RDF triple is deleted from the database, the corresponding link can be safely removed. However, this does not necessarily translate to removing nodes at the same time. A node cannot be removed if there is at least one link pointing to it.
Support for RDF Queries

At the SQL level, a new table function called RDF_MATCH is used to provide the query interface for the RDF model. This table function RDF_MATCH can be used to query the RDF data stored in the database and it can support searching for an arbitrary pattern against the RDF data.

A SQL query to search against the RDF data looks like this:

```sql
SELECT t.GetSubject
FROM TABLE (RDF_MATCH (RDFModels('Reviewers'), RDFPattern (RDF_Triple('?r', 'ReviewerOf', '?c'), RDF_Triple('?r', 'rdf:type', 'Ph. D. Student'), RDF_Triple('?r', 'age', '?a'), '?a < 25'))) t;
```

Assuming there is a model called “Reviewers” that has data for all the reviewers for a conference, this query can find all the reviewers who are Ph.D. students and have age less than 25 years.