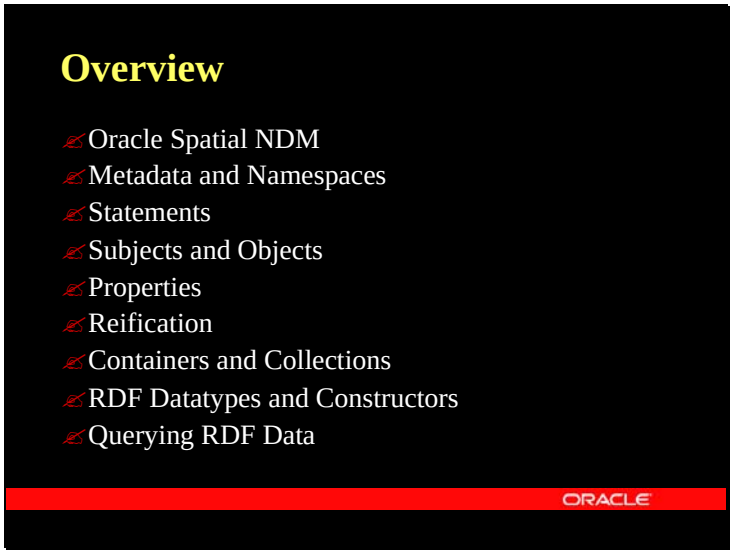
Slide 1

RDF in Oracle Spatial

This paper is based on Oracle 10g support for RDF data storage in Oracle Spatial.

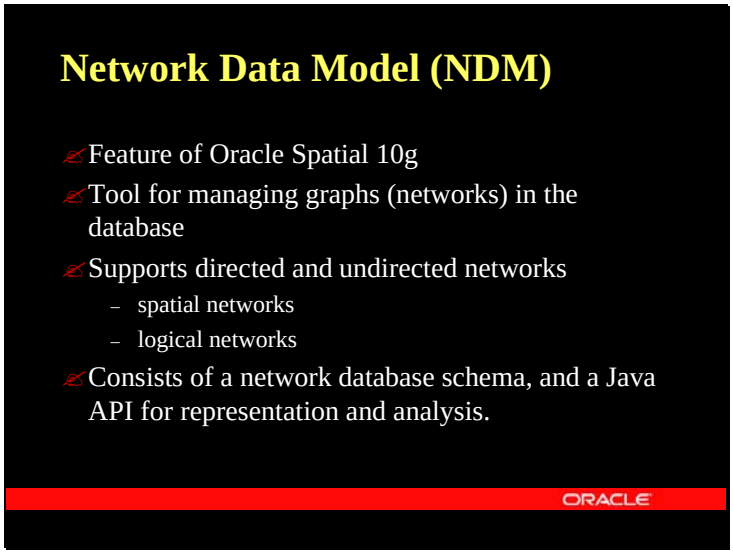
Slide 2



Overview

This presentation is a high-level overview of RDF data storage in the Oracle Spatial Network Data Model.
I will first introduce the Oracle Spatial Network Data Model product, and then describe how it has been extended to store RDF-modeled data.
At the end of this presentation you will have a good idea of how RDF data is handled in an Oracle database.

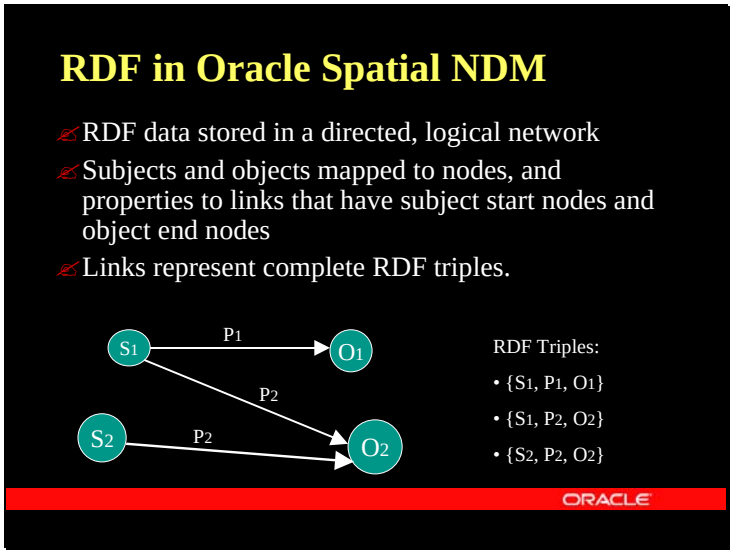
Slide 3



Network Data Model (NDM)

Oracle Spatial is an option of the Oracle database, and the Network Data Model is one of the many features provided with Oracle Spatial 10g.
A network or graph captures relationships between objects using connectivity.
NDM supports both directed and undirected networks, which can be either spatial or logical.
Spatial networks contain both connectivity information and geometric information.
Logical Networks contain connectivity information but no geometric information.
NDM consists of two components: a network database schema, and a Java API.
The network schema contains network metadata and tables for nodes and links.
The Java API enables network representation and network analysis.

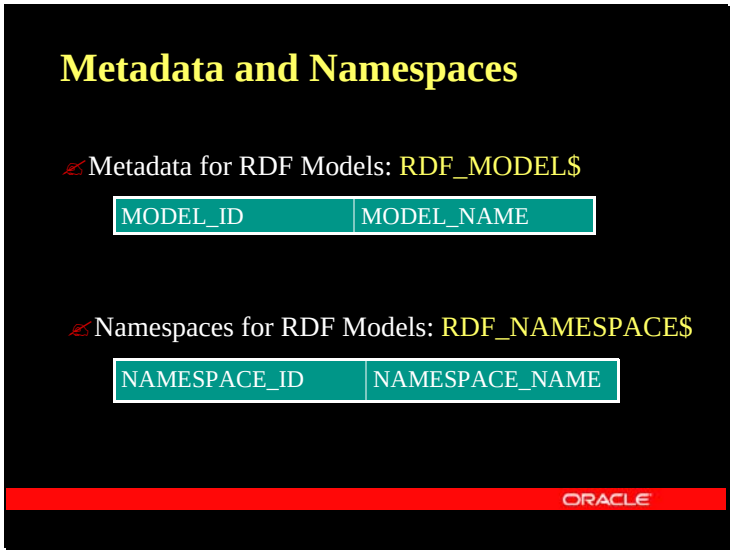
Slide 4



RDF in Oracle Spatial NDM

NDM stores RDF data using a directed, logical network.
Generally speaking, NDM maps subjects and objects of statements to nodes in a network and properties to links. In NDM, nodes are stored in a system nodes' table and links in a system links' table. Each link must have a start node and and end node. For RDF storage, the start node of a link is the subject of a statement, and the end node of a link is the object of a statement.
A link therefore represents a complete RDF triple.
A key feature of RDF storage in NDM is that subject and object nodes are stored only once, regardless of the number of times they participate in triples.
Subject and object nodes are reused, if they already exist in the database.
A new link, however, is always created whenever a new triple is inserted.
When a triple is deleted from the database, the corresponding link is directly removed. However, the nodes attached to this link are not removed if there is at least one other link connected to them.
This is the basic storage model for RDF-modeled data. The rest of the presentation reviews the enhancements to NDM to fully support the management of RDF-modeled data.

Slide 5

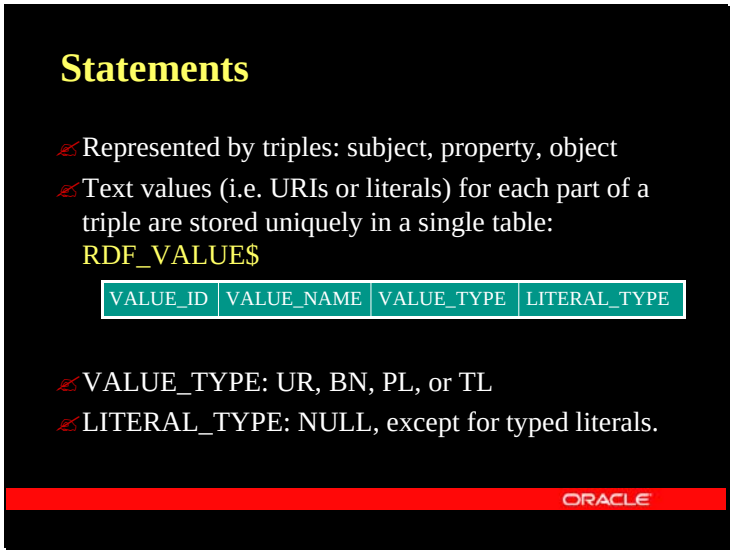


Metadata and Namespaces

Metadata for RDF Models
RDF_MODEL$ is a system level table created to store information on all the models defined in the database.
When a new RDF model is created, an entry is made to this table.
The MODEL_ID is automatically generated and can be used instead of the model's name to refer to a particular model.

Namespaces for RDF Models
Namespaces are used in RDF/XML documents to make these documents readable.
In NDM, namespaces are stored directly with their subject, properties, and objects. However, this table can be optionally used to catalog all the namespaces used in an RDF universe.

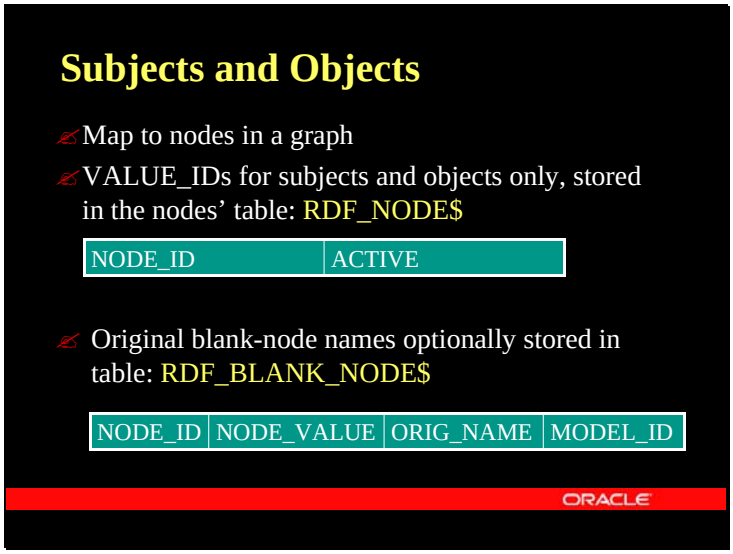
Slide 6



Statements

Statements are represented by triples: subject, property, and object.
RDF_VALUE$ stores the text values, i.e. the URIs or literals for each part of a triple.
Each text value is stored only once, and a unique VALUE_ID is generated for the text entry.
Possible VALUE_TYPE entries are URIs, blank nodes, plain literals or typed literals.
Only typed literals will have a LITERAL_TYPE entry.

Slide 7

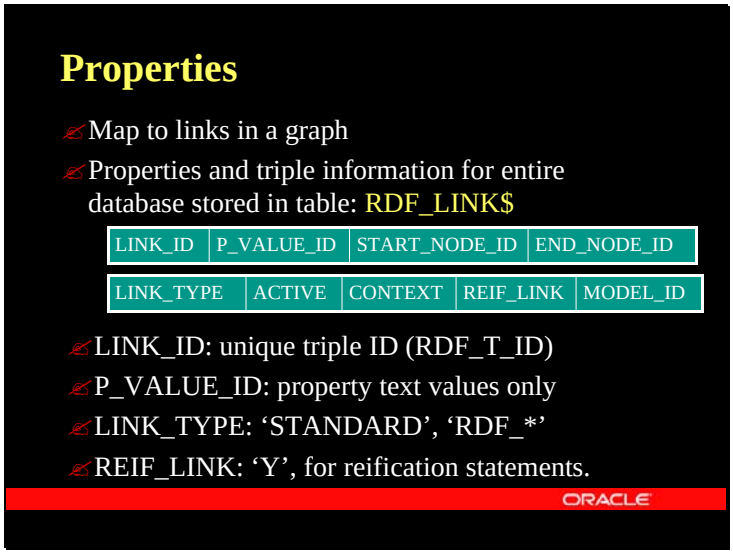


Subjects and Objects

RDF_NODE$ stores only the VALUE_ID for text values that participate in subjects or objects of statements.
The NODE_ID is the same as the VALUE_ID.
NODE_ID values are stored only once, regardless of the number of subjects or objects they participate in.
The nodes' table allows RDF data to be exposed to all the analytical functions and APIs available in core NDM.

Blank Nodes
Blank nodes are used to represent unknown nodes. They are also used when the relationship between a subject node and an object node is n-ary (as is the case with containers).
New blank nodes are automatically generated whenever blank nodes are encountered in triples.
However, the user has the option to reuse a particular blank node. This is necessary when inserting containers and collections.
RDF_BLANK_NODE$ stores the original names of blank nodes that are to be reused when encountered in triples.

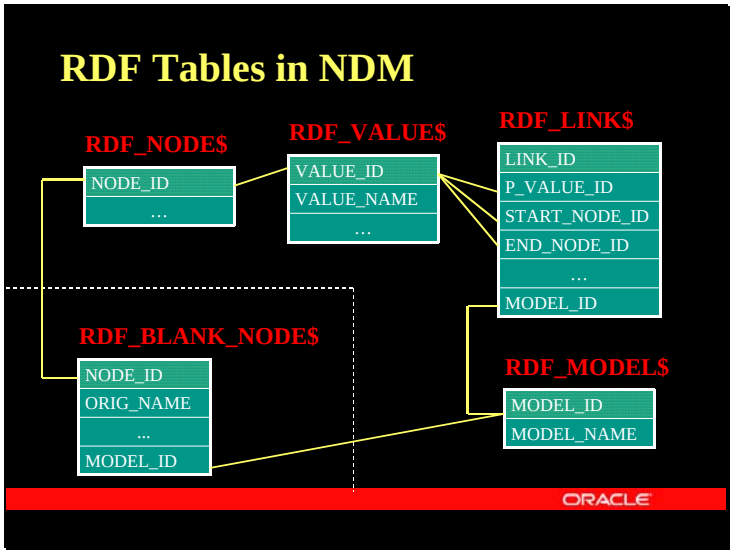
Slide 8



Properties

RDF_LINK$ stores all the triples for all the RDF models in the database. The MODEL_ID logically partitions the RDF_LINK$ table. Selecting all the links for a specified MODEL_ID, returns the RDF network for that particular model.

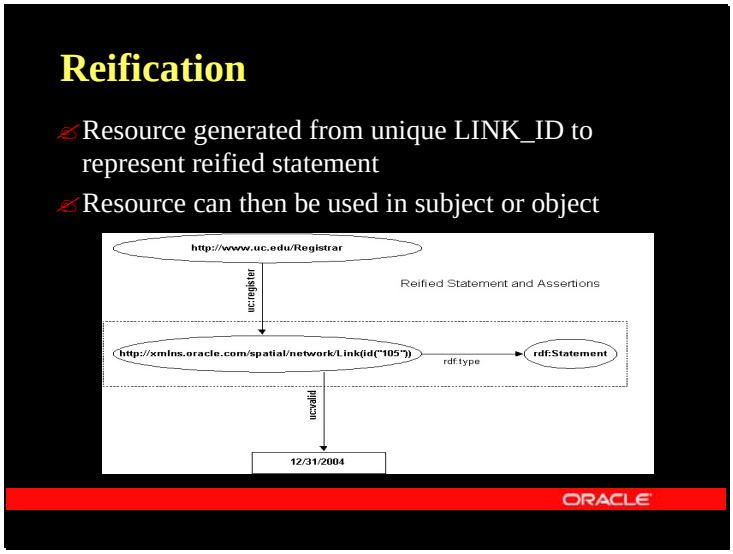
Slide 9



RDF Tables in NDM

RDF_VALUE$, RDF_NODE$, RDF_LINK$, and RDF_MODEL$ are the key tables for RDF storage. RDF_BLANK_NODE$ is used when reusing blank nodes, and RDF_NAMESPACE$ is used for cataloging purposes only.

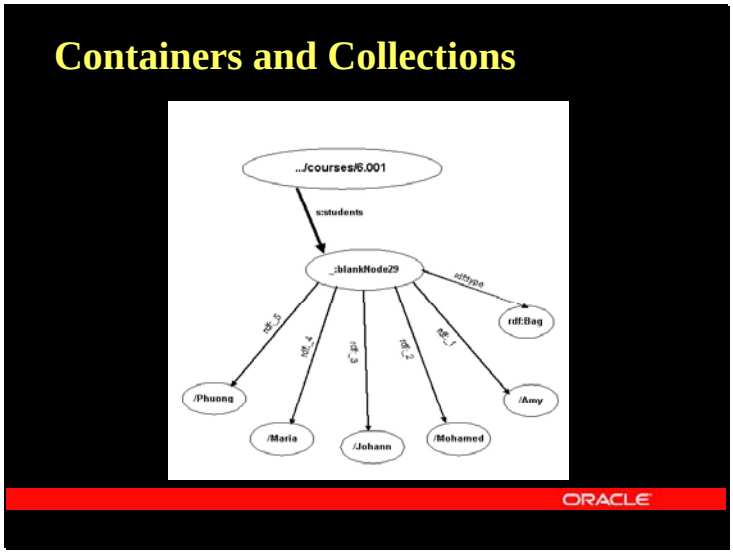
Slide 10



Reification

A reification of a statement in RDF is a description of the statement using an RDF statement.
To represent a reified statement in NDM a resource is generated using the triple's LINK_ID (RDF_T_ID). This resource can then be used as the subject or object of a statement.
To process a reification statement, a triple is first entered with the reified statement's resource as subject, rdf:type as property and rdf:Statement as object.
A triple is then entered for each assertion about the reified statement.
Each reified statement will have only one rdf:type –> rdf:Statement associated with it, regardless of the number of assertions made using this resource.

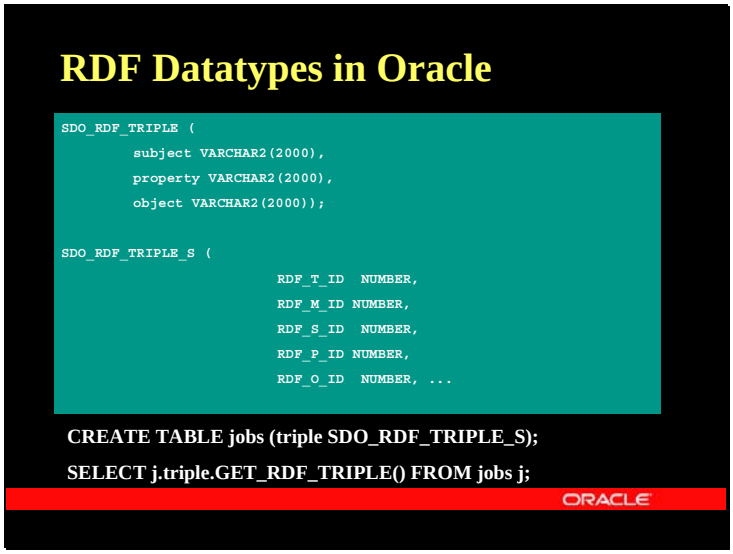
Slide 11



Containers and Collections

Containers and collections are handled similarly in NDM.
Each container or collection will have a rdf:type -> rdf:container_name/collection_name associated with it.
The LINK_TYPE for container or collection members are RDF_MEMBER.
Collections have an additional constraint: no new entries can be added to the list.

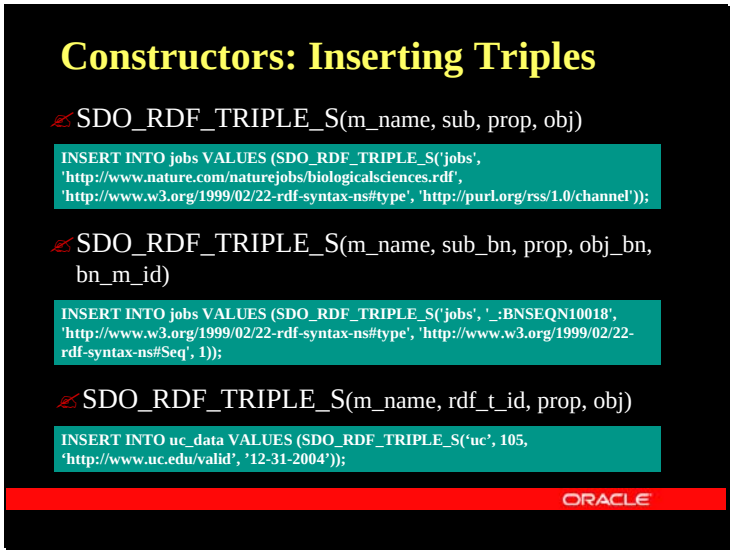
Slide 12



RDF Datatypes in Oracle

Two new datatypes are defined for RDF-modeled data:
The SDO_RDF_TRIPLE type is defined to serve as the triple representation of RDF data.
The SDO_RDF_TRIPLE_S type is defined to store persistent data in the database.
The GET_RDF_TRIPLE() function returns an SDO_RDF_TRIPLE type.

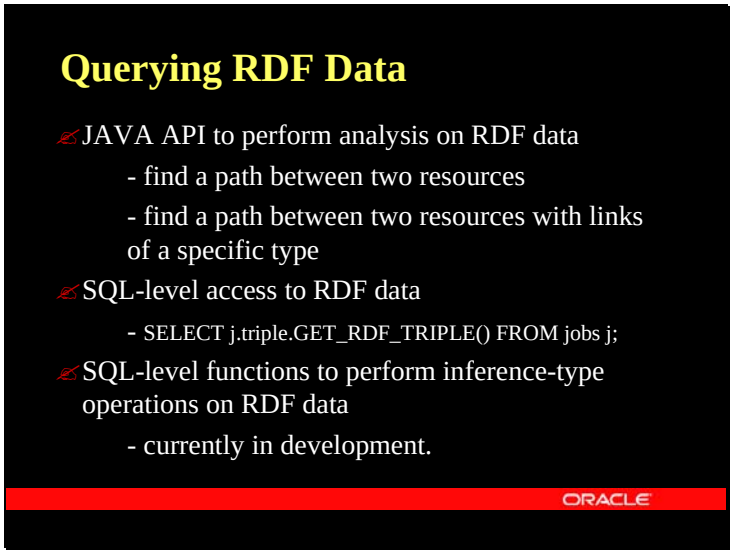
Slide 13



Constructors: Inserting RDF Triples

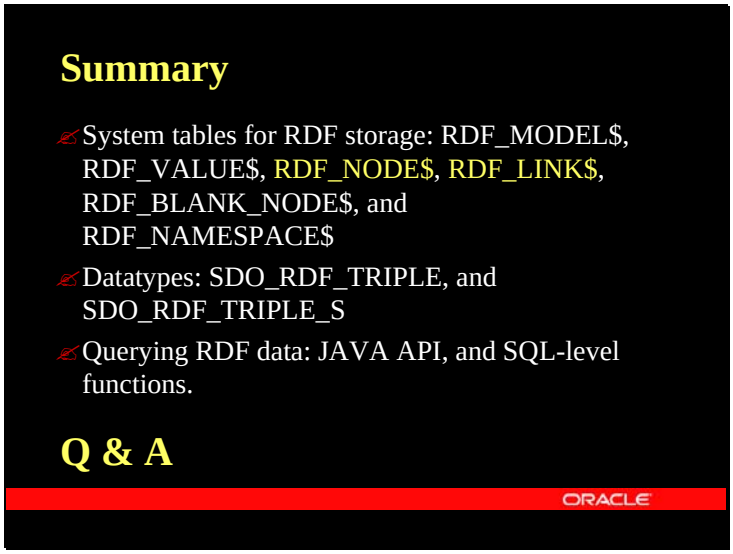
General constructor for triple insertion: SDO_RDF_TRIPLE_S(model_name, subject, property, object).
Constructor for reusing blank nodes: SDO_RDF_TRIPLE_S(model_name, sub_or_bn, property, obj_or_bn, bn_m_id). This constructor is required for entering containers and collections.
Constructor for reifying statements: SDO_RDF_TRIPLE_S(model_name, rdf_t_id, property, object).

Slide 14



Querying RDF Data

Slide 15



Summary

There are six system tables for RDF data storage in NDM. Two of the tables: RDF_NODE$ and RDF_LINK$ are required tables for NDM, the other 4 tables have been added to support RDF data storage.
Two new datatypes are defined for RDF-modeled data: one for representing triples, the other for storing triples.
JAVA API and SQL-level functions for querying RDF data.

Q&A