Get the most from Oracle Database 11g
Semantic Technology: Best Practices

Xavier Lopez, Ph.D., Director, Oracle Server Technologies
Souripriya Das, Ph.D., Consultant Member, Oracle Server Technologies
Outline

• Background
• Oracle 11g DB Semantic Technologies
• Enhancements in latest release of 11g
• Best Practices
• Performance
• Summary
Introduction to Oracle Semantic Technology
Semantic Data Management Characteristics

- Discovery of data relationships across...
  - Structured data (database, apps, web services)
  - Unstructured data (email, office documents) Multi-data types (graphs, spatial, text, sensors)
- Text Mining & Web Mining infrastructure
  - Terabytes of structured & unstructured data
- Queries are not defined in advance
- Schemas are continuously evolving
- Associate more meaning (context) to enterprise data to enable its (re)use across applications
- Allow sharing and reuse of enterprise and web data.
- Built on open, industry W3C standards:
  - SQL, XML, RDF, OWL, SPARQL
Text Mining: National Intelligence

Information Extraction
Categorization, Feature/term Extraction

Processed Document Collection
RDF/OWL

Ontology Engineering Modeling Process
Ontologies
Domain Specific Knowledge Base

Web Resources
News, Email, RSS
Content Mgmt. Systems

Analyst
Explore
SQL/SPARQL Query

Browsing, Presentation, Reporting, Visualization, Query
Data Integration Platform in Health Informatics

Enterprise Information Consumers (EICs):
- Patient Care
- Workforce Management
- Business Intelligence
- Clinical Analytics

Run-Time Metadata

Integration Server (Semantic Knowledge base)

Access

Deploy

Model
- Virtual

Relate

Model
- Physical
Data Integration Platform in Health Informatics

Enterprise Information Consumers (EICs)

Run-Time Metadata
Access

Integration Server (Semantic Knowledge base)

Access
Deploy
Model Virtual
Relate
Model Physical

Design-Time Metadata

Patient Care
Workforce Management
Business Intelligence
Clinical Analytics

LIS
CIS
HTB
HIS
Semantic Data Management Workflow

Edit & Transform
- Entity Extraction & Transform
- Ontology Engineering
- Categorization
- Custom Scripting

Load, Query & Inference
- RDF/OWL Data Management
- SQL & SPARQL Query
- Inferencing
- Semantic Rules
- Scalability & Security

Applications & Analysis
- Graph Visualization
- Link Analysis
- Statistical Analysis
- Faceted Search
- Pattern Discovery
- Text Mining

Data Sources
- Transaction Systems
- Unstructured Content
- RSS, email
- Other Data Formats

Partners

Partners
Oracle 11g RDF/OWL Graph Data Management

- Oracle 11g is the leading commercial database with native RDF/OWL data management
- Scalable & secure platform for wide-range of semantic applications
- Readily scales to ultra-large repositories (+1 billion)
- Choice of SQL or SPARQL query
- Leverages Oracle Partitioning and Advanced Compression. RAC supported
- Growing ecosystem of 3rd party tools partners

Key Capabilities:

Load / Storage
- Native RDF graph data store
- Manages billions of triples
- Fast batch, bulk and incremental load

Query
- SQL: SEM_Match
- SPARQL: via Jena plug-in
- Ontology assisted query of RDBMS data

Reasoning
- Forward chaining model
- RDFS++, OWL, OWLPrime
- User defined rule base
Commitment to W3C Semantic Standards

• Our implementation entirely based on W3C standards (RDF, RDFS, OWL)
  • SPARQL support through Jena
• Members of following W3C Web Semantic Activities:
  • W3C Data Access Working Group (DAWG)
  • W3C OWL Working group
  • W3C Semantic Web Education & Outreach (SWEO)
  • W3C Health Care & Life Sciences Interest Group (HCLS)
  • W3C Multimedia Semantics Incubator group
  • W3C Semantic Web Rules Language (SWRL)
Semantic Technology Ecosystem
Integrated Tools and Solution Providers:

- TopQuadrant
- protégé
- Boeing
- Raytheon
- Metatomix
- Mondeca
- Cognia
- Siderean
- CollabRx
-Franz Inc.
- Ontoprise
-MedTrust
- Teranode
-Saltlux
-Yena
-Orbis Technologies, Inc.
DB Semantic Technology in Oracle 11g
Technical Overview
The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle’s products remains at the sole discretion of Oracle.
Semantic Technology: Building Blocks

• Representation
  • RDF (Resource Description Framework)
  • Vocabularies
    • RDFS (RDF Schema Language)
    • OWL (Web Ontology Language)

• Inference
  • Implicit rules that capture semantics of each vocabulary
    • RDF, RDFS, OWL-Lite, OWL-DL, OWL-Full
  • User-defined rules that goes beyond the standard vocabularies

• Query
  • Using graph-patterns in languages such as SPARQL
Example: Store, Infer, Query
Example: Store, Infer, Query

:California :partOf :USA :partOf :NorthAmerica

:partOf rdf:type owl:TransitiveProperty
Example: Store, Infer, Query

Asserted Facts

<table>
<thead>
<tr>
<th>:partOf</th>
<th>rdf:type</th>
<th>owl:TransitiveProperty</th>
</tr>
</thead>
<tbody>
<tr>
<td>:California</td>
<td>:partOf</td>
<td>:USA</td>
</tr>
<tr>
<td>:USA</td>
<td>:partOf</td>
<td>:NorthAmerica</td>
</tr>
<tr>
<td>:partOf</td>
<td>rdf:type</td>
<td>owl:TransitiveProperty</td>
</tr>
<tr>
<td>:partOf</td>
<td>:partOf</td>
<td>:NorthAmerica</td>
</tr>
</tbody>
</table>
Example: Store, Infer, Query

Assorted Facts

<table>
<thead>
<tr>
<th>:partOf</th>
<th>rdf:type</th>
<th>owl:TransitiveProperty</th>
</tr>
</thead>
<tbody>
<tr>
<td>:California</td>
<td>:partOf</td>
<td>:USA</td>
</tr>
<tr>
<td>:USA</td>
<td>:partOf</td>
<td>:NorthAmerica</td>
</tr>
<tr>
<td>:partOf</td>
<td>rdf:type</td>
<td>owl:TransitiveProperty</td>
</tr>
</tbody>
</table>
Example: Store, Infer, Query

Asserted Facts

<table>
<thead>
<tr>
<th>:partOf</th>
<th>rdf:type</th>
<th>owl:TransitiveProperty</th>
</tr>
</thead>
<tbody>
<tr>
<td>:California</td>
<td>:partOf</td>
<td>:USA</td>
</tr>
<tr>
<td>:USA</td>
<td>:partOf</td>
<td>:NorthAmerica</td>
</tr>
</tbody>
</table>

Derived Facts

<table>
<thead>
<tr>
<th>:California</th>
<th>:partOf</th>
<th>:NorthAmerica</th>
</tr>
</thead>
</table>
Example: Store, Infer, Query

**Asserted Facts**

<table>
<thead>
<tr>
<th>:partOf</th>
<th>rdf:type</th>
<th>owl:TransitiveProperty</th>
</tr>
</thead>
<tbody>
<tr>
<td>:California</td>
<td>:partOf</td>
<td></td>
</tr>
<tr>
<td>:USA</td>
<td>:partOf</td>
<td></td>
</tr>
<tr>
<td>:NorthAmerica</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Derived Facts**

<table>
<thead>
<tr>
<th>:California</th>
<th>:partOf</th>
<th>:NorthAmerica</th>
</tr>
</thead>
</table>

**Query:**

```
SELECT ?x ?y
FROM ...
WHERE { ?x :partOf ?y }
```
Example: Store, Infer, Query

Asserted Facts

<table>
<thead>
<tr>
<th>:partOf</th>
<th>rdf:type</th>
<th>owl:TransitiveProperty</th>
</tr>
</thead>
<tbody>
<tr>
<td>:California</td>
<td>:partOf</td>
<td></td>
</tr>
<tr>
<td>:USA</td>
<td>:partOf</td>
<td></td>
</tr>
<tr>
<td>:NorthAmerica</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Derived Facts

<table>
<thead>
<tr>
<th>:California</th>
<th>:partOf</th>
<th>:NorthAmerica</th>
</tr>
</thead>
</table>

Query: SELECT ?x ?y
FROM ... WHERE { ?x :partOf ?y }

Result: ?x  ?y
         :California :USA
         :California :NorthAmerica
         :USA        :NorthAmerica
Oracle 11g DB Semantic Technologies

- Storage model, loading, and management for data represented in RDF/OWL
- SQL-based query of RDF/OWL data
- Ontology-assisted query of Relational data
- Native inferencing engine to infer new relationships from RDF/OWL data
- Java-based API
Functionality: Overview

- **INFER**
  - RDF/S
  - User-def.

- **QUERY**
  - Query RDF/OWL data and ontologies

- **STORE**
  - RDF/OWL data and ontologies & rulebases
  - Enterprise (Relational) data
  - Batch-Load
  - Incr. DML
  - Bulk-Load
Functionality: Overview

**INFER**
- OWL subsets
- RDF/S
- User-def.

**QUERY**
- Query RDF/OWL data and ontologies

**STORE**
- Incremental (Incr.)
- Batch Load
- User-defined

RDF/OWL data and ontologies & rulebases

Enterprise (Relational) data
Functionality: Overview

- **INFER**
  - OWL subsets
  - RDF/S
  - User-def.

- **QUERY**
  - Query RDF/OWL data and ontologies
  - Ontology-Assisted Query of Enterprise Data

- **STORE**
  - Incr. DML
  - Batch-Load
  - Bulk-Load

- **RDF/OWL data and ontologies & rulebases**
- **Enterprise (Relational) data**
Storage: Overview

Application Tables with RDF object type columns

Models:
- Model 1
- Model 2
- Model n

RDF/OWL data and ontologies

Vocabularies and Rulebases
- OWL subset
- RDF / RDFS
- Rulebase m

Inferred Triple Sets:
- Inferred Triple Set 1
- Inferred Triple Set 2
- Inferred Triple Set p

Rules Indexes (Derived data)

Application Tables with RDF object type columns:
- A₁, A₂, ..., Aₙ

R HERMAN SCOTT
R SCOTT
Storage: Overview

Oracle DB Semantic Network (inside MDSYS)

Application Tables with RDF object type columns

RDF/OWL data and ontologies

Vocabularies and Rulebases

OWL subset

RDF / RDFS

Rulebase m

Inferred Triple Set 1

Inferred Triple Set 2

Inferred Triple Set p

Rules Indexes (Derived data)

Model 1

Model 2

Model n

A1

A2

An

A1

A2

An

R

R

R

...
Major Steps for building semantic app.
Major Steps for building semantic app.

- Create **Semantic Network** in Oracle
Major Steps for building semantic app.

- Create Semantic Network in Oracle
- Create an RDF/OWL model associating with a table column of type SDO_RDF_TRIPLE_S
Major Steps for building semantic app.

- Create Semantic Network in Oracle
- Create an RDF/OWL model associating with a table column of type SDO_RDF_TRIPLE_S
- Optionally, create user-defined rulebases
Major Steps for building semantic app.

- Create **Semantic Network** in Oracle
- Create an **RDF/OWL model** associating with a table column of type SDO_RDF_TRIPLE_S
- Optionally, create user-defined rulebases
- Store new RDF/OWL data into Oracle via bulk-load, batch-load, or SQL INSERT; Also, perform any **DML**
Major Steps for building semantic app.

- Create **Semantic Network** in Oracle
- Create an **RDF/OWL model** associating with a table column of type SDO_RDF_TRIPLE_S
- Optionally, create user-defined rulebases
- **Store** new RDF/OWL data into Oracle via bulk-load, batch-load, or SQL INSERT; Also, perform any **DML**
- **Infer** new RDF/OWL data using Oracle’s native inference engine
Major Steps for building semantic app.

- Create **Semantic Network** in Oracle
- Create an **RDF/OWL model** associating with a table column of type SDO_RDF_TRIPLE_S
- Optionally, create user-defined rulebases
- Store new RDF/OWL data into Oracle via bulk-load, batch-load, or SQL INSERT; Also, perform any **DML**
- **Infer** new RDF/OWL data using Oracle’s native inference engine
- **Query** RDF/OWL data and ontologies via SQL using SEM_MATCH table function; optionally **combine with SQL operations** such as join, order by, group by, filter conditions, etc.
Major Steps for building semantic app.

- Create **Semantic Network** in Oracle
- Create an **RDF/OWL model** associating with a table column of type `SDO_RDF_TRIPLE_S`
- Optionally, create user-defined rulebases
- **Store** new RDF/OWL data into Oracle via bulk-load, batch-load, or SQL INSERT; Also, perform any **DML**
- **Infer** new RDF/OWL data using Oracle’s native inference engine
- **Query** RDF/OWL data and ontologies via SQL using `SEM_MATCH` table function; optionally **combine with SQL operations** such as join, order by, group by, filter conditions, etc.
- Perform **Ontology-assisted Query** against enterprise (relational) data using `SEM_RELATED` and `SEM_DISTANCE` operators
Creating Semantic Network and Semantic Models
Creating Semantic Network and Semantic Models

Creating a semantic network

1. \texttt{SEM\_APIS.CREATESEM\_NETWORK(<tablespace>);}
Creating Semantic Network and Semantic Models

Creating a semantic network
1. SEM_APIS.CREATE_SEM_NETWORK (<tablespace>);

Creating a semantic model
2. Create an application table with an SDO_RDF_TRIPLE_S type col
   CREATE TABLE ATAB (ID int, TRI SDO_RDF_TRIPLE_S) compress;

3. Create a model associated with the SDO_RDF_TRIPLE_S column
   SEM_APIS.CREATE_SEM_MODEL (  
     'MODEL1',  
     'ATAB',  
     'TRI'  
   );
Loading RDF/OWL data

• Load data using
  • **Bulk-load** (fastest, but skips triples containing long literals)
    • Load data into a staging table (using SQL*Loader from a file or Named Pipe containing N-Triple formatted data)
    • Invoke `sem_apis.bulk_load_from_staging_table` to do bulk load from the staging table
  • **Batch-load** (fast, can handle long literals as well)
    • Use the `oracle.spatial.rdf.client.BatchLoader` class packaged in `<ORACLE_HOME>/md/jlib/sdordf.jar` to load from file containing N-Triple formatted data
  • **SQL INSERT** (for loading small amounts of data)
Query RDF Data

- SPARQL-like graph pattern embedded in SQL query
- Matches RDF/OWL graph patterns with patterns in stored data
- Returns a table of results
- Can use SQL operators/functions to process results
- Avoids staging when combined with queries on relational data

```sql
SELECT ...
FROM ..., TABLE (SEM_MATCH invocation) t, ...
WHERE ...
```
Query RDF Data

- SPARQL-like graph pattern embedded in SQL query
- Matches RDF/OWL graph patterns with patterns in stored data
- Returns a *table* of results
- Can use SQL operators/functions to process results
- Avoids staging when combined with queries on relational data

SELECT ...
FROM ...
TABLE (SEM_MATCH invocation) t, ...
WHERE ...

```sql
SEM_MATCH (  
  '(?x :partOf :NorthAmerica)',   -- pattern: all persons
  SEM_Models('gmap'),            -- RDF/OWL data models
  SEM_Rulebases('OWLPRIME'),     -- rulebases
  SEM_Aliases(...)              -- aliases
  null                          -- no filter condition
  NULL                          -- (optional) index_status
  NULL                          -- (optional) options
)  
```
Query RDF Data

- SPARQL-like graph pattern embedded in SQL query
- Matches RDF/OWL graph patterns with patterns in stored data
- Returns a table of results
- Can use SQL operators/functions to process results
- Avoids staging when combined with queries on relational data

```
SELECT ...
FROM ...,
TABLE (SEM_MATCH invocation) t, ...
WHERE ...
```
Table Columns returned by SEM_MATCH

Each returned row contains one (or more) of the following cols for each variable $x$ in the graph-pattern:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>varchar2</td>
<td>Value matched with $x$</td>
</tr>
<tr>
<td>x$rdfVTYP</td>
<td>varchar2</td>
<td>Value Type: URI, Literal, or Blank Node</td>
</tr>
<tr>
<td>x$rdfLTYP</td>
<td>varchar2</td>
<td>Literal Type: e.g., xsd:integer</td>
</tr>
<tr>
<td>x$rdfCLOB</td>
<td>CLOB</td>
<td>CLOB value matched with $x$</td>
</tr>
<tr>
<td>x$rdfLANG</td>
<td>varchar2</td>
<td>Language tag: e.g., “en-us”</td>
</tr>
</tbody>
</table>

**Projection Optimization**: Only the columns referred to by the containing query are returned.
Multi-model RDF Query in SQL
Multi-model RDF Query in SQL

:OracleHQemployee :HQLoc :California

:John

emp

erdf:type
Multi-model RDF Query in SQL

- :OracleHQemployee rdf:type :John
- :HQLoc :California
- :USA :partOf :NorthAmerica
- :partOf :partOf

emp
gmap
Multi-model RDF Query in SQL

:OracleHQemployee rdf:type :John

:HQLoc :California

:partOf :USA :partOf :NorthAmerica
SELECT e, e$rdfvtyp, p, p$rdfvtyp FROM
TABLE(SEM_MATCH(
  (?e rdf:type ?empCategory)
  (?empCategory :HQLoc ?loc)
  (?loc :partOf ?p),
  SEM_Models('emp', 'gmap'), SEM_Rulebases('OWLPRIME'),
  SEM_ALIASES(SEM_ALIAS('', 'http://example.org/')),NULL));
Multi-model RDF Query in SQL

```
SELECT e, e$rdfvtyp, p, p$rdfvtyp FROM TABLE(SEM_MATCH('(?e rdf:type ?empCategory)
(?empCategory :HQLoc ?loc)
(?loc :partOf ?p)'), SEM_Models('emp', 'gmap'), SEM_Rulebases('OWLPRIME'),
SEM_ALIASES(SEM_ALIAS('', 'http://example.org/')),NULL));
```

<table>
<thead>
<tr>
<th>e</th>
<th>e$rdfvtyp</th>
<th>p</th>
<th>p$rdfvtyp</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://example.org/John">http://example.org/John</a></td>
<td>URI</td>
<td><a href="http://example.org/USA">http://example.org/USA</a></td>
<td>URI</td>
</tr>
<tr>
<td><a href="http://example.org/John">http://example.org/John</a></td>
<td>URI</td>
<td><a href="http://example.org/NorthAmerica">http://example.org/NorthAmerica</a></td>
<td>URI</td>
</tr>
</tbody>
</table>
Ontology-Assisted Query: Overview

• Motivation
  • Traditionally relationship between two terms is checked only in a syntactic manner
  • Need a new operator which can do semantic relationship check by consulting an ontology

• Introduces two operators
  • SEM_RELATED (<col>,<pred>, <ontologyTerm>, <ontologyName> [,<invoc_id>])
  • SEM_DISTANCE (<invoc_id>) ← Ancillary Oper.
Example: Query using Semantic Operators

```
<table>
<thead>
<tr>
<th>ID</th>
<th>DIAGNOSIS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Hand_Fracture</td>
</tr>
<tr>
<td>2</td>
<td>Rheumatoid_Arthritis</td>
</tr>
</tbody>
</table>
```

“Find all entries in diagnosis column that are related to ‘Upper_Extremity_Fracture’”

Syntactic match returns no rows:
```
SELECT p_id, diagnosis FROM Patients WHERE diagnosis = 'Upper_Extremity_Fracture';
```
Example: Query using Semantic Operators

```sql
SELECT p_id, diagnosis
FROM Patients
WHERE SEM_RELATED (diagnosis, 'rdfs:subClassOf', 'Upper_Extremity_Fracture', 'Medical_ontology') = 1;
```
Example: Query using Semantic Operators

SELECT p_id, diagnosis FROM Patients
WHERE SEMRELATED(
  diagnosis,
  ‘rdfs:subClassOf’,
  ‘Upper_Extremity_Fracture’,
  ‘Medical_ontology’, 123) = 1
AND SEMDISTANCE(123) <= 2;

<table>
<thead>
<tr>
<th>ID</th>
<th>DIAGNOSIS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Hand_Fracture</td>
</tr>
<tr>
<td>2</td>
<td>Rheumatoid_Arthritis</td>
</tr>
</tbody>
</table>
Example: Query using Semantic Operators

```
Patients
<table>
<thead>
<tr>
<th>ID</th>
<th>DIAGNOSIS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Hand_Fracture</td>
</tr>
<tr>
<td>2</td>
<td>Rheumatoid_Arthritis</td>
</tr>
</tbody>
</table>
```

“Find all entries in diagnosis column that are related to ‘Upper_Extremity_Fracture’”

Syntactic match returns no rows:
```
SELECT p_id, diagnosis FROM Patients WHERE diagnosis = ‘Upper_Extremity_Fracture’;
```
Inference: Overview

- Native inferencing for
  - OWL subsets (including OWLPrime)
  - RDF, RDFS
  - User-defined rules
- RDF/OWL graph is *entailed* (new triples are inferred) by applying rules in rulebase(s) to model(s)
- Inferencing is based on forward chaining: new triples are inferred and stored ahead of query time
Enhancements in the latest 11g release

• Query performance
  • Flexible Indexing to customize set of B-Tree indexes on a model
  • Hint framework to allow custom hints
  • Additional cols returned by SEM.MATCH: id, prefix, suffix

• Bulk-load performance
  • Custom hints for faster bulk-load
  • Bulk-append performance enhancements
  • Parallelism within a bulk-load or bulk-append session

• Stats collection
  • New procedures for collecting stats for a model or rules index
Canonical vs. Exact Lexical Form of Literal values

[simplified description for clarity]

- Inserting the following triple into an empty app. Table
  - `<xyz.com/John> <xyz.com/age> “032”^^xsd:decimal`

adds the following rows to the Oracle Semantic store.
Canonical vs. Exact Lexical Form of Literal values

[simplified description for clarity]

• Inserting the following triple into an empty app. Table
  • `<xyz.com/John> <xyz.com/age> “032”^^xsd:decimal`
  adds the following rows to the Oracle Semantic store

RDF_VALUE$
Canonical vs. Exact Lexical Form of Literal values

[simplified description for clarity]

• Inserting the following triple into an empty app. Table
  • `<xyz.com/John>`  `<xyz.com/age>`  “032”^^xsd:decimal

adds the following rows to the Oracle Semantic store

<table>
<thead>
<tr>
<th>Value_id</th>
<th>Vname_prefix</th>
<th>Vname_suffix</th>
<th>Value_type</th>
<th>Literal_type</th>
<th>Canon_id</th>
</tr>
</thead>
</table>
| RDF_VALUE$
Canonical vs. Exact Lexical Form of Literal values

[simplified description for clarity]

- Inserting the following triple into an empty app. Table
  - `<xyz.com/John> <xyz.com/age> "032"^^xsd:decimal`
  adds the following rows to the Oracle Semantic store

<table>
<thead>
<tr>
<th>Value_id</th>
<th>Vname_prefix</th>
<th>Vname_suffix</th>
<th>Value_type</th>
<th>Literal_type</th>
<th>Canon_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>id1</td>
<td>xyz.com/</td>
<td>John</td>
<td>UR</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>
Canonical vs. Exact Lexical Form of Literal values

[simplified description for clarity]

- Inserting the following triple into an empty app. Table
  - `<xyz.com/John> <xyz.com/age> “032”^^xsd:decimal`
  
  adds the following rows to the Oracle Semantic store

<table>
<thead>
<tr>
<th>Value_id</th>
<th>Vname_prefix</th>
<th>Vname_suffix</th>
<th>Value_type</th>
<th>Literal_type</th>
<th>Canon_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>id1</td>
<td>xyz.com/</td>
<td>John</td>
<td>UR</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>id2</td>
<td>xyz.com/</td>
<td>age</td>
<td>UR</td>
<td>NULL</td>
<td>NULL</td>
</tr>
</tbody>
</table>
Canonical vs. Exact Lexical Form of Literal values

[simplified description for clarity]

- Inserting the following triple into an empty app. Table
  - `<xyz.com/John> <xyz.com/age> “032”^^xsd:decimal`
  adds the following rows to the Oracle Semantic store

<table>
<thead>
<tr>
<th>Value_id</th>
<th>Vname_prefix</th>
<th>Vname_suffix</th>
<th>Value_type</th>
<th>Literal_type</th>
<th>Canon_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>id1</td>
<td>xyz.com/</td>
<td>John</td>
<td>UR</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>id2</td>
<td>xyz.com/</td>
<td>age</td>
<td>UR</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>id3</td>
<td>32</td>
<td>NULL</td>
<td>TL</td>
<td>xsd:decimal</td>
<td>NULL</td>
</tr>
</tbody>
</table>
Canonical vs. Exact Lexical Form of Literal values

[simplified description for clarity]

- Inserting the following triple into an empty app. Table
  - `<xyz.com/John>  <xyz.com/age>  “032”^^xsd:decimal`
  adds the following rows to the Oracle Semantic store

<table>
<thead>
<tr>
<th>Value_id</th>
<th>Vname_prefix</th>
<th>Vname_suffix</th>
<th>Value_type</th>
<th>Literal_type</th>
<th>Canon_id</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td>id1</td>
<td>xyz.com/</td>
<td>John</td>
<td>UR</td>
<td>NULL</td>
<td>NULL</td>
<td>…</td>
</tr>
<tr>
<td>id2</td>
<td>xyz.com/</td>
<td>age</td>
<td>UR</td>
<td>NULL</td>
<td>NULL</td>
<td>…</td>
</tr>
<tr>
<td>id3</td>
<td>32</td>
<td>NULL</td>
<td>TL</td>
<td>xsd:decimal</td>
<td>NULL</td>
<td>…</td>
</tr>
<tr>
<td>id4</td>
<td>032</td>
<td>NULL</td>
<td>TL</td>
<td>xsd:decimal</td>
<td>id3</td>
<td>…</td>
</tr>
</tbody>
</table>
Canonical vs. Exact Lexical Form of Literal values

[simplified description for clarity]

• Inserting the following triple into an empty app. Table
  • `<xyz.com/John>`  `<xyz.com/age>`  “032”^^xsd:decimal
    adds the following rows to the Oracle Semantic store

<table>
<thead>
<tr>
<th>Value_id</th>
<th>Vname_prefix</th>
<th>Vname_suffix</th>
<th>Value_type</th>
<th>Literal_type</th>
<th>Canon_id</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td>id1</td>
<td>xyz.com/</td>
<td>John</td>
<td>UR</td>
<td>NULL</td>
<td>NULL</td>
<td>…</td>
</tr>
<tr>
<td>id2</td>
<td>xyz.com/</td>
<td>age</td>
<td>UR</td>
<td>NULL</td>
<td>NULL</td>
<td>…</td>
</tr>
<tr>
<td>id3</td>
<td>32</td>
<td>NULL</td>
<td>TL</td>
<td>xsd:decimal</td>
<td>NULL</td>
<td>…</td>
</tr>
<tr>
<td>id4</td>
<td>032</td>
<td>NULL</td>
<td>TL</td>
<td>xsd:decimal</td>
<td>id3</td>
<td>…</td>
</tr>
</tbody>
</table>

RDF Model
Canonical vs. Exact Lexical Form of Literal values

[simplified description for clarity]
- Inserting the following triple into an empty app. Table:
  - `<xyz.com/John> <xyz.com/age> “032”^^xsd:decimal`
  adds the following rows to the Oracle Semantic store:

<table>
<thead>
<tr>
<th>Value_id</th>
<th>Vname_prefix</th>
<th>Vname_suffix</th>
<th>Value_type</th>
<th>Literal_type</th>
<th>Canon_id</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>id1</td>
<td>xyz.com/</td>
<td>John</td>
<td>UR</td>
<td>NULL</td>
<td>NULL</td>
<td>...</td>
</tr>
<tr>
<td>id2</td>
<td>xyz.com/</td>
<td>age</td>
<td>UR</td>
<td>NULL</td>
<td>NULL</td>
<td>...</td>
</tr>
<tr>
<td>id3</td>
<td>32</td>
<td>NULL</td>
<td>TL</td>
<td>xsd:decimal</td>
<td>id3</td>
<td>...</td>
</tr>
<tr>
<td>id4</td>
<td>032</td>
<td>NULL</td>
<td>TL</td>
<td>xsd:decimal</td>
<td>id3</td>
<td>...</td>
</tr>
</tbody>
</table>

RDF Model:

| Model_id | Start_Node_Id | P_Value_Id | Canon_End_Node_Id | End_Node_Id | ... |
Canonical vs. Exact Lexical Form of Literal values

[simplified description for clarity]

- Inserting the following triple into an empty app. Table
  - `<xyz.com/John>  <xyz.com/age>  “032”^^xsd:decimal`
  
  adds the following rows to the Oracle Semantic store

<table>
<thead>
<tr>
<th>Value_id</th>
<th>Vname_prefix</th>
<th>Vname_suffix</th>
<th>Value_type</th>
<th>Literal_type</th>
<th>Canon_id</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td>id1</td>
<td>xyz.com/</td>
<td>John</td>
<td>UR</td>
<td>NULL</td>
<td>NULL</td>
<td>…</td>
</tr>
<tr>
<td>id2</td>
<td>xyz.com/</td>
<td>age</td>
<td>UR</td>
<td>NULL</td>
<td>NULL</td>
<td>…</td>
</tr>
<tr>
<td>id3</td>
<td>32</td>
<td>NULL</td>
<td>TL</td>
<td>xsd:decimal</td>
<td>NULL</td>
<td>…</td>
</tr>
<tr>
<td>id4</td>
<td>032</td>
<td>NULL</td>
<td>TL</td>
<td>xsd:decimal</td>
<td>id3</td>
<td>…</td>
</tr>
</tbody>
</table>

RDF Model

<table>
<thead>
<tr>
<th>Model_id</th>
<th>Start_Node_Id</th>
<th>P_Value_Id</th>
<th>Canon_End_Node_Id</th>
<th>End_Node_Id</th>
<th>…</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>id1</td>
<td>id2</td>
<td>id3</td>
<td>id4</td>
<td>…</td>
</tr>
</tbody>
</table>
Canonical vs. Exact Lexical Form of Literal values

[simplified description for clarity]

- Inserting the following triple into an empty app. Table
  - `<xyz.com/John> <xyz.com/age> “032”^^xsd:decimal`
  adds the following rows to the Oracle Semantic store

<table>
<thead>
<tr>
<th>Value_id</th>
<th>Vname_prefix</th>
<th>Vname_suffix</th>
<th>Value_type</th>
<th>Literal_type</th>
<th>Canon_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>id1</td>
<td>xyz.com/</td>
<td>John</td>
<td>UR</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>id2</td>
<td>xyz.com/</td>
<td>age</td>
<td>UR</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>id3</td>
<td>32</td>
<td>NULL</td>
<td>TL</td>
<td>xsd:decimal</td>
<td>NULL</td>
</tr>
<tr>
<td>id4</td>
<td>032</td>
<td>NULL</td>
<td>TL</td>
<td>xsd:decimal</td>
<td>id3</td>
</tr>
</tbody>
</table>
Canonical vs. Exact Lexical Form of Literal values

[simplified description for clarity]

- Inserting the following triple into an empty app. Table
- `<xyz.com/John> <xyz.com/age> “032”^^xsd:decimal`

adds the following rows to the Oracle Semantic store

<table>
<thead>
<tr>
<th>Value_id</th>
<th>Vname_prefix</th>
<th>Vname_suffix</th>
<th>Value_type</th>
<th>Literal_type</th>
<th>Canon_id</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>id1</td>
<td>xyz.com/</td>
<td>John</td>
<td>UR</td>
<td>NULL</td>
<td>NULL</td>
<td>...</td>
</tr>
<tr>
<td>id2</td>
<td>xyz.com/</td>
<td>age</td>
<td>UR</td>
<td>NULL</td>
<td>NULL</td>
<td>...</td>
</tr>
<tr>
<td>id3</td>
<td>32</td>
<td>NULL</td>
<td>TL</td>
<td>xsd:decimal</td>
<td>NULL</td>
<td>...</td>
</tr>
<tr>
<td>id4</td>
<td>032</td>
<td>NULL</td>
<td>TL</td>
<td>xsd:decimal</td>
<td>id3</td>
<td>...</td>
</tr>
</tbody>
</table>

RDF Model

<table>
<thead>
<tr>
<th>Model_id</th>
<th>Start_Node_Id</th>
<th>P_Value_Id</th>
<th>Canon_End_Node_Id</th>
<th>End_Node_Id</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>id1</td>
<td>id2</td>
<td>id3</td>
<td>id4</td>
<td>...</td>
</tr>
</tbody>
</table>
Default Indexes in Oracle Sem. Store

- Indexes on RDF Model:
  - Unique index: \( <P,C,S,M> \) where
    - \( P \rightarrow P\_VALUE\_ID \) (predicate id)
    - \( C \rightarrow CANON\_END\_NODE\_ID \) (canonical object id)
    - \( S \rightarrow START\_NODE\_ID \) (subject id)
    - \( M \rightarrow MODEL\_ID \) (model id)
  - Non-unique index: \( <P,S,C,F> \) where
    - \( P, S, \) and \( C \rightarrow \) same as above
    - \( F \rightarrow (\text{case when canon}\_\text{end}\_\text{node}\_\text{id}=\text{end}\_\text{node}\_\text{id} \text{then null else end}\_\text{node}\_\text{id} \text{end}) \)

- Indexes on RDF\_VALUE$:
  - Unique index: \( <\text{value}\_\text{id}> \)
  - Unique index: \( <\text{vname}\_\text{prefix}, \text{value}\_\text{type}, \text{vname}\_\text{suffix}, \text{literal}\_\text{type}, \text{language}\_\text{type}> \)
Prefix and Suffix of URIs
Prefix and Suffix of URIs

- For URIs, *suffix* is the portion of *value_name* to the right of the rightmost occurrence of ‘#’, ’/’, or ‘:’. If suffix is too long, then it is NULL. The rest is the *prefix*.
  - Example: http://purl.uniprot.org/uniprot/Q4U9M9
- For all other types of RDF terms, suffix is NULL and prefix is the whole *value_name*. 
Prefix and Suffix of URIs

• For URIs, suffix is the portion of value_name to the right of the rightmost occurrence of ‘#’, ’/’, or ‘:’. If suffix is too long, then it is NULL. The rest is the prefix.
  • Example: http://purl.uniprot.org/uniprot/Q4U9M9
• For all other types of RDF terms, suffix is NULL and prefix is the whole value_name.
• For faster lookup against a large RDF_VALUE$ table, use:
  • select * from mdsys.rdf_value$ where
    vname_prefix=sem_apis.value_name_prefix('http://purl.uniprot.org/uniprot/Q4U9M9','UR')
    and value_type='UR'
    and
    vname_suffix=sem_apis.value_name_suffix('http://purl.uniprot.org/uniprot/Q4U9M9','UR');
Prefix and Suffix of URIs

- For **URIs**, *suffix* is the portion of *value_name* to the right of the rightmost occurrence of ‘#’, ‘/’, or ‘:’. If suffix is too long, then it is NULL. The rest is the *prefix*.
  - Example: *http://purl.uniprot.org/uniprot/Q4U9M9*
- For all other types of RDF terms, suffix is NULL and prefix is the whole *value_name*.
- For faster lookup against a large RDF_VALUE$ table, use:
  - select * from mdsys.rdf_value$ where *vname_prefix*=sem_apis.value_name_prefix(
    'http://purl.uniprot.org/uniprot/Q4U9M9','UR')
  and *value_type*='UR'
  and
  *vname_suffix*=sem_apis.value_name_suffix(
    'http://purl.uniprot.org/uniprot/Q4U9M9','UR');
  instead of:
  - select * from mdsys.rdf_value$ where *value_name*='http://purl.uniprot.org/uniprot/Q4U9M9';
Each returned row contains one (or more) of the following cols for each variable \( ?x \) in the graph-pattern:

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x )</td>
<td>varchar2</td>
<td>Value matched with ( ?x )</td>
</tr>
<tr>
<td>( x$rdfvid)</td>
<td>number</td>
<td>Value ID for matching value</td>
</tr>
<tr>
<td>( x$_prefix)</td>
<td>varchar2</td>
<td>PREFIX of the matched value</td>
</tr>
<tr>
<td>( x$_suffix)</td>
<td>varchar2</td>
<td>SUFFIX of the matched value</td>
</tr>
<tr>
<td>( x$rdfVTYP)</td>
<td>varchar2</td>
<td>Value TYPE: URI, Literal, or Blank Node</td>
</tr>
<tr>
<td>( x$rdfLTYP)</td>
<td>varchar2</td>
<td>Literal TYPE: e.g., xsd:integer</td>
</tr>
<tr>
<td>( x$rdfCLOB)</td>
<td>CLOB</td>
<td>CLOB value matched with ( ?x )</td>
</tr>
<tr>
<td>( x$rdfLANG)</td>
<td>varchar2</td>
<td>LANGUAGE tag: e.g., “en-us”</td>
</tr>
</tbody>
</table>

**Projection Optimization**: Only the columns referred to by the containing query are returned.
Flexible Indexing

- Allows custom B-Tree indexing for RDF models and rules indexes

- `add_sem_index (index_code)`
  - Adds a new nonunique index for every RDF model
  - status of the indexes is ‘UNUSABLE’
  - Index_code is some combination of one or more of the column abbreviations: S,P,C,O,F. (Example: ‘CPS’)

- `alter_index_on_sem_graph(graph_name,index_code, command, … is_rules_index)`
  - Rebuilds or marks as UNUSABLE the index on the specified RDF model or rules index

- `drop_sem_index (index_code)`
HINT0 framework: basic syntax

• Query hints may be specified using the HINT0 framework as part of the `options` parameter of SEM_MATCH table function

• Syntax
  • `SEM_MATCH(… options => ‘… HINT0 = { … } … ’ …)`
HINT0 Framework: using vid to avoid join
HINT0 Framework: using vid to avoid join

- If HINT0 is used, and, for a variable in the pattern, if the containing SQL query references *only* its vid column, then join with RDF_VALUE$ for that variable is avoided.
HINT0 Framework: using vid to avoid join

- If HINT0 is used, and, for a variable in the pattern, if the containing SQL query references only its vid column, then join with RDF_VALUE$ for that variable is avoided.

```sql
select /*+ leading(t) use_nl(v) */
    v.value_name group_name, t.member_count
from (
    SELECT y$rdfvid, count(*) member_count
    FROM table(sem_match(
        '(?x rdf:type ?y)',
        SEM_Models('rdf_model_uniprot'),null,null,null,
        options => 'HINT0={}'
    ))
    GROUP BY y$rdfvid
) t, mdsys.rdf_value$ v
where t.y$rdfvid = v.value_id;
```
HINT0 framework: Aliases
SELECT e, p FROM
TABLE(SEM_MATCH(‘(?e rdf:type ?empCategory)
(?empCategory :HQLoc ?loc)
(?loc :partOf ?p)’,
SEM_Models(‘emp’, ‘gmap’), SEM_Rulebases(‘OWLPRIME’),
SEM_ALIASES(SEM_ALIAS(‘’, 'http://example.org/')),NULL));
HINT0 framework: Aliases

```sql
SELECT e, p FROM
  TABLE(SEM_MATCH('(?e rdf:type ?empCategory)
              (?empCategory :HQLoc ?loc)
              (?loc :partOf ?p'),
        SEM_Models('emp', 'gmap'), SEM_Rulebases('OWLPRIME'),
        SEM_ALIASES(SEM_ALIAS('', 'http://example.org/')), NULL));
```
HINT0 framework: Aliases

```
SELECT e, p FROM TABLE(SEM_MATCH(  
  (?e rdf:type ?empCategory)  
  (?empCategory :HQLoc ?loc)  
  (?loc :partOf ?p),  
  SEM_Models('emp', 'gmap'), SEM_Rulebases('OWLPRIME'),  
  SEM_ALIASES(SEM_ALIAS('', 'http://example.org/')),NULL));
```
SELECT e, p FROM TABLE(SEM_MATCH('(?e rdf:type ?empCategory) (?empCategory :HQLoc ?loc) (?loc :partOf ?p)'), SEM_Models('emp', 'gmap'), SEM_Rulebases('OWLPRIME'), SEM_ALIASES(SEM_ALIAS('', 'http://example.org/')), NULL);
HINT0 framework: Aliases

SELECT e, p FROM
TABLE(SEM_MATCH(
  "(?e rdf:type ?empCategory)"
  "(?empCategory :HQLoc ?loc)"
  "(?loc :partOf ?p)"
),
SEM_Models('emp', 'gmap'), SEM_Rulebases('OWLPRIME'),
SEM_ALIASES(SEM_ALIAS('', 'http://example.org/')),NULL);
HINT0 framework: Aliases

SELECT e, p FROM TABLE(SEM_MATCH( (?e rdf:type ?empCategory) (?empCategory :HQLoc ?loc) (?loc :partOf ?p)), SEM_Models('emp', 'gmap'), SEM_Rulebases('OWLPRIME'), SEM_ALIASES(SEM_ALIAS('', 'http://example.org/')), NULL));

- Aliases for the tables involved in the Join (in translated SQL query)
  - For the 3 triple patterns: t0, t1, t2 \(\rightarrow\) aliases for RDF Model
  - For the 4 distinct variables: ?e, ?empCategory, ?loc, ?p \(\rightarrow\) aliases for RDF_VALUE$

- 7-way Join query:
  - t0 \(\times\) t1 \(\times\) t2 \(\times\) ?e \(\times\) ?empCategory \(\times\) ?loc \(\times\) ?p

- Use of *projection optimization* reduces it to a 5-way join query since columns for only the ?e and ?p variables are referred in containing SQL query: t0 \(\times\) t1 \(\times\) t2 \(\times\) ?e \(\times\) ?p
HINT0 framework: Hints

- Various kinds of hints may be specified via HINT0
  - SQL hints
    - Join order: LEADING(...), ORDERED
    - Join type: USE_HASH(...), USE_NL(...), ...
    - Access path: INDEX(...), ...
  - Special hints
    - Canonical value: GET_CANON_VALUE(…)
      - If the specified variables need to be retrieved, then always retrieves their canonical lexical forms (not their exact lexical forms). (The wildcard ?* may be used to indicate all variables.)
  - Aliases
    - Variable names: ?x, ?z, ...
    - Pattern ordinal based: t0, t1, t2, ...

Example: Using Hint0 and _prefix column

SELECT count(*), avg(length(protein)), avg(length(author)), avg(length(title))
FROM TABLE( SEM_MATCH ( 
    '(?protein rdf:type :Protein)
    (?protein :citation ?citation)
    (?citation :author ?author)
    (?citation :title ?title)',
    SEM_Models('rdf_model_uniprot'),NULL,
    SEM_ALIASES(SEM_ALIAS('', 'http://purl.uniprot.org/core/'))
    ,null
    ,options => 'HINT0={LEADING(?author t2)
      USE_NL(t0 t1 t2 t3 ?protein ?author ?title)
      GET_CANON_VALUE(?*)}'
))
where author$_prefix like 'Bairoch%';
ALLOW_DUP=T for multi-model queries

- ALLOW_DUP=T may be specified as part of the options parameter in SEM_MATCH involving multiple models
  - SEM_MATCH(… options => ‘… ALLOW_DUP=T … ’ …)
- This avoids elimination of duplicates in the combined content of those models
- Leads to significant performance benefits
Bulk-load: IZC, MBV, and MBT steps

1. load External File

2. Select distinct values

3a. Detect and Resolve collisions

3b. Merge batch values

4. Create All collisions table

5. Generate and insert Id-based triples

6. Generate and insert Id-based triples

7. Insert + index build OR Append with duplicate removal

Global

Local
Bulk-load: IZC, MBV, and MBT steps

1. **External File**
   - Load

2. **Staging Table**
   - Select distinct values

3a. **IZC**
   - Detect and resolve collisions

3b. **LexValues Table**
   - Merge batch values

4. **AllCollisionExt Table**
   - Create all collisions table

5. **BatchIdTriples Table**
   - Generate and insert Id-based triples

6. **BatchLexValues Table**
   - Select distinct values

7. **IdTriples Table**
   - Insert + index build OR Append with duplicate removal

   Remove duplicate triples

Global

Local
Bulk-load: IZC, MBV, and MBT steps

1. Load
   - External File

2. Staging Table
   - Select distinct values

3a. IZC
   - Detect and Resolve collisions

3b. MBV
   - Merge batch values

4. AllCollisionExt Table
   - Create All collisions table

5. BatchIdTriples Table
   - Generate and insert Id-based triples

6. BatchLexValues Table
   - Generate and insert batch values

7. IdTriples Table
   - Insert + index build OR Append with duplicate removal

   Remove duplicate triples

Global

Local
Bulk-load: IZC, MBV, and MBT steps

1. load
2. Select distinct values
3a. IZC Detect and Resolve collisions
3b. MBV Merge batch values
4. Create All collisions table
5. BatchIdTriples Table Generate and insert Id-based triples
6. BatchLexValues Table Select distinct values
7. IdTriples Table Insert + index build OR Append with duplicate removal
Global
MBT
Local
External File
Bulk-Load / Bulk-Append: Enhancements

- The "flags" parameter in `sem_apis.bulk_load_from_staging_table` may be used for hints
  - Join type hint: `USE_HASH`, `USE_NL`, ...
    - `IZC_JOIN_HINT` (IZC → Is_Zero_Collisions step)
    - `MBV_JOIN_HINT` (MBV → Merge_Batch_Values step)
    - `MBT_JOIN_HINT` (MBT → Merge_Batch_Triples step)
    - Example: `flags => ' ... MBT_JOIN_HINT=USE_NL ...'`
- Parallelism within a bulk-load or bulk-append
  - Example: `flags => ' ... PARALLEL=4 ... '`
- Optimistic bulk-append
  - Assumes overlap between new batch and current content less likely
    - `MBT_PIECEMEAL` hint helps in case of overlap
  - Staging table does not need the *_ext columns
  - Event traces from multiple bulk-loads can now be recorded in a single table, `RDF$ET_TAB`, in the invoker's schema
Stats Collection for model or rules index

- Allows users to collect stats on a
  - Model
    - `sem_apis.analyze_model(model_name)`
  - Rules index
    - `sem_apis.analyze_rules_index(index_name)`
- Only the owner or dba can use these procedures
Best Practices: Query
Best Practices: Query

• **LEADING** hint may be used to specify a selectivity-based (prefix or full) **ordering of triple-patterns and variables**
Best Practices: Query

- **LEADING** hint may be used to specify a selectivity-based (prefix or full) **ordering of triple-patterns and variables**
- Using `GET_CANON_VALUE(?)` in HINT0
Best Practices: Query

- **LEADING** hint may be used to specify a selectivity-based (prefix or full) ordering of triple-patterns and variables
- Using `GET_CANON_VALUE(?)` in HINT0
- Retrieving only the `<var>$RDFVID` col, when possible
Best Practices: Query

- **LEADING** hint may be used to specify a selectivity-based (prefix or full) **ordering of triple-patterns and variables**
- Using `GET_CANON_VALUE(?)` in HINT0
- Retrieving only the `<var>$RDFVID` col, when possible
- Using `<var>$_prefix col based filter` instead of `<var>`
Best Practices: Query

- **LEADING** hint may be used to specify a selectivity-based (prefix or full) ordering of triple-patterns and variables
- Using **GET_CANON_VALUE(?*)** in HINT0
- Retrieving only the `<var>$RDFVID` col, when possible
- Using `<var>$_prefix col` based filter instead of `<var>`
- Using **ALLOW_DUP=T** as hint in multi-model queries
  - Note: The result table may contain additional duplicates
- **USE_NL** for variables helps when number of values expected to be retrieved (via ID → Value mapping) is not too high. Otherwise, **USE_HASH** may be used for the variables.
Best Practices: Query
Best Practices: Query

- Creating **function-based indexes** on RDF_VALUE$ based upon functions matching selective filters frequently encountered in a query workload may help performance.
  - Example: creating a date index on RDF_VALUE$
- Creating **new indexes** (such as, ‘CPS’, ‘CS’, or ‘S’) on the RDF models based upon query workload may lead to faster response time.
Best Practices: Bulk-Load
Best Practices: Bulk-Load

- For smaller bulk-load or bulk-append, optionally, [IZC|MBV|MBT]_JOIN_HINT=USE_NL may be used
Best Practices: Bulk-Load

- For **smaller** bulk-load or bulk-append, optionally, 
  \[IZC|MBV|MBT\]_JOIN_HINT=USE_NL may be used
- For **larger** bulk-loads or bulk-appends, USE_HASH join hint may provide faster completion.
Best Practices: Bulk-Load

• For smaller bulk-load or bulk-append, optionally, [IZC|MBV|MBT]_JOIN_HINT=USE_NL may be used.
• For larger bulk-loads or bulk-appends, USE_HASH join hint may provide faster completion.
• For larger bulk-appends with possible overlap with pre-existing model data, MBT_PIECEMEAL may help.
Best Practices: Bulk-Load

- For smaller bulk-load or bulk-append, optionally, [IZC|MBV|MBT]_JOIN_HINT=USE_NL may be used.
- For larger bulk-loads or bulk-appends, USE_HASH join hint may provide faster completion.
- For larger bulk- appends with possible overlap with pre-existing model data, MBT_PIECEMEAL may help.
- For larger bulk-loads or bulk-appends on multi-CPU machines, following options may allow faster load:
  - PARALLEL=n
  - PARALLEL_CREATE_INDEX
# Performance Factors

<table>
<thead>
<tr>
<th>Bulk-Load Performance</th>
<th>Factors affecting Performance</th>
<th>Oracle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration</td>
<td>Desktop vs. Server</td>
<td>Desktop Machine</td>
</tr>
<tr>
<td>Indexes</td>
<td>Number of Indexes</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>Type of Indexes</td>
<td>B-tree indexes</td>
</tr>
<tr>
<td></td>
<td>Columns Indexed</td>
<td>PCSM and PSCF (C: canon-obj-id col) (F: C=O ? NULL : 0)</td>
</tr>
<tr>
<td>Parallelism</td>
<td></td>
<td>Not used in these benchmark tests</td>
</tr>
</tbody>
</table>
Desktop Performance: Settings

- **Hardware**
  - CPU → Single-CPU P4 (3.0GHz with Hyper Threading)
  - Memory → 4GB
  - Hard Disks → Two 500GB 7200rpm SATA 3.0G
- **OS:** Red Hat Enterprise Linux (32-bit)
- **DBMS**
  - Oracle DBMS Enterprise Server 11g Release 1
  - Settings
    - db_block_size=8192
    - pga_aggregate_target=2000M
    - sga_target=1800M
    - Db_file_multiblock_read_count=128
    - Filesystemio_options=‘SETALL’
  - Temp tablespace was allocated on a separate hard disk
Performance: Load and Inference
Performance: Load and Inference

• Bulk-Load
  • 1.1 billion triples (LUBM8000)
    • Time for loading staging table ➔ 2 ½ hrs (strict parsing: 11 ½ hrs)
    • Time for Bulk-load API ➔ about 31 hrs
  • Storage ➔ data 41 GB, indexes 94 GB, app table 22 GB

• Inference using OWLPrime
  • 1.068 billion triples (LUBM8000 minus the duplicate triples)
    • Inferred triples ➔ 521.7 million
    • Time ➔ 56.7 hrs
Performance: Query (new UniProt data)

<table>
<thead>
<tr>
<th>Description</th>
<th>Pattern</th>
<th>Projection</th>
<th>Result limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1: Display the ranges of transmembrane regions</td>
<td>6 triples 5 vars</td>
<td>3 vars</td>
<td>15000 rows</td>
</tr>
<tr>
<td>Q2: List proteins with publications by authors with matching names</td>
<td>5 triples 5 vars 1 LIKE pred.</td>
<td>3 vars</td>
<td>10 rows</td>
</tr>
<tr>
<td>Q3: Count the number of times a publication by a specific author is cited</td>
<td>3 triples 2 vars</td>
<td>0 vars</td>
<td>32 rows</td>
</tr>
<tr>
<td>Q4: List resources that are related to proteins annotated with a specific keyword</td>
<td>3 triples 2 vars</td>
<td>1 var</td>
<td>3000 rows</td>
</tr>
<tr>
<td>Q5: List genes associated with human diseases</td>
<td>7 triples 5 vars</td>
<td>3 vars</td>
<td>750 rows</td>
</tr>
<tr>
<td>Q6: List recently modified entries</td>
<td>2 triples 2 vars 1 range pred.</td>
<td>2 vars</td>
<td>8000 rows</td>
</tr>
</tbody>
</table>
## Performance: Query (new UniProt data)

### Table 4. Queries adapted from UniProt sample queries

<table>
<thead>
<tr>
<th>Description</th>
<th>Pattern</th>
<th>Projection</th>
<th>Result limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1: Display the ranges of transmembrane regions</td>
<td>6 triples</td>
<td>3 vars</td>
<td>15000 rows</td>
</tr>
<tr>
<td></td>
<td>5 vars</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q2: List proteins with publications by authors</td>
<td>5 triples</td>
<td>3 vars</td>
<td>10 rows</td>
</tr>
<tr>
<td>with matching names</td>
<td>5 vars</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 LIKE pred.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q3: Count the number of times a publication by</td>
<td>3 triples</td>
<td>0 vars</td>
<td>32 rows</td>
</tr>
<tr>
<td>a specific author is cited</td>
<td>2 vars</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q4: List resources that are related to proteins</td>
<td>3 triples</td>
<td>1 var</td>
<td>3000 rows</td>
</tr>
<tr>
<td>annotated with a specific keyword</td>
<td>2 vars</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q5: List genes associated with human diseases</td>
<td>7 triples</td>
<td>3 vars</td>
<td>750 rows</td>
</tr>
<tr>
<td></td>
<td>5 vars</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q6: List recently modified entries</td>
<td>2 triples</td>
<td>2 vars</td>
<td>8000 rows</td>
</tr>
<tr>
<td></td>
<td>2 vars</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 range pred.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

80 M triples
### Performance: Query (new UniProt data)

<table>
<thead>
<tr>
<th>Description</th>
<th>Pattern</th>
<th>Projection</th>
<th>Result limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1: Display the ranges of transmembrane regions</td>
<td>6 triples</td>
<td>3 vars</td>
<td>15,000 rows</td>
</tr>
<tr>
<td></td>
<td>5 vars</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q2: List proteins with publications by authors with matching names</td>
<td>5 triples</td>
<td>3 vars</td>
<td>10 rows</td>
</tr>
<tr>
<td></td>
<td>5 vars</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 LIKE pred.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q3: Count the number of times a publication by a specific author is cited</td>
<td>3 triples</td>
<td>0 vars</td>
<td>32 rows</td>
</tr>
<tr>
<td></td>
<td>2 vars</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q4: List resources that are related to proteins annotated with a specific keyword</td>
<td>3 triples</td>
<td>1 var</td>
<td>3,000 rows</td>
</tr>
<tr>
<td></td>
<td>2 vars</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q5: List genes associated with human diseases</td>
<td>7 triples</td>
<td>3 vars</td>
<td>750 rows</td>
</tr>
<tr>
<td></td>
<td>5 vars</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q6: List recently modified entries</td>
<td>2 triples</td>
<td>2 vars</td>
<td>8,000 rows</td>
</tr>
<tr>
<td></td>
<td>2 vars</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 range pred.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Performance: Query (new UniProt data)

Table 4. Queries adapted from UniProt sample queries

<table>
<thead>
<tr>
<th>Description</th>
<th>Pattern</th>
<th>Projection</th>
<th>Result limit</th>
<th>#Result rows</th>
<th>Time (in sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1: Display the ranges of transmembrane regions</td>
<td>6 triples 5 vars</td>
<td>3 vars</td>
<td>15000 rows</td>
<td>0</td>
<td>10.01</td>
</tr>
<tr>
<td>Q2: List proteins with publications by authors with matching names</td>
<td>5 triples 5 vars 1 LIKE pred.</td>
<td>3 vars</td>
<td>10 rows</td>
<td>521</td>
<td>0.03</td>
</tr>
<tr>
<td>Q3: Count the number of times a publication by a specific author is cited</td>
<td>3 triples 2 vars</td>
<td>0 vars</td>
<td>32 rows</td>
<td>578</td>
<td>0.02</td>
</tr>
<tr>
<td>Q4: List resources that are related to proteins annotated with a specific keyword</td>
<td>3 triples 2 vars</td>
<td>1 var</td>
<td>3000 rows</td>
<td>2,413</td>
<td>0.04</td>
</tr>
<tr>
<td>Q5: List genes associated with human diseases</td>
<td>7 triples 5 vars</td>
<td>3 vars</td>
<td>750 rows</td>
<td>7,812</td>
<td>0.53</td>
</tr>
<tr>
<td>Q6: List recently modified entries</td>
<td>2 triples 2 vars 1 range pred.</td>
<td>2 vars</td>
<td>8000 rows</td>
<td>6,626,072</td>
<td>206.14</td>
</tr>
</tbody>
</table>

(uses date index)
Summary

- Oracle 11g DB Semantic Technologies provides following support for RDF/OWL
  - Storage
  - Loading
  - Inference
  - Querying
  - that is
  - efficient
  - scales to more than billion triples
  - with latest release of 11g further allowing fine tuning of
  - Query: via hints framework and flexible indexing
  - Bulk Load and Bulk Append: via new flags
For More Information

search.oracle.com

semantic technologies

or

oracle.com
Related Open World activities

• Monday Develop Keynote
  • Ted Farrell, SVP/Chief Architect, Developer Tools "Breaking Enterprise Application Platform Barriers"
    (Mon. Sept. 22, 10:15am - Salon 9)
  • Arrive at 9:00 for Charles Phillips / Chuck Rozwat Video feed
• Tuesday Develop Keynote
  • Tom Kyte, Senior Database Architect "The Best Way..."
    (Tues. Sept. 23, 10:15am – Salon 9)
### Semantics at OOW 2008 - Sessions

<table>
<thead>
<tr>
<th>Date/Time</th>
<th>Title</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sunday, Sept 21</td>
<td>Get the Most from Oracle Database 11g Semantic Technology: Best Practices</td>
<td>Marriott Salon 12/13</td>
</tr>
<tr>
<td>1:15-2:15 p.m.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Monday, Sept 22</td>
<td>Using RDFS/OWL for Building Semantic Web Applications</td>
<td>Marriott Salon 12/13</td>
</tr>
<tr>
<td>1:00-2:00 p.m.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4:00-5:00 p.m.</td>
<td>Oracle Database 11g Production Case Study: Teranode and Pfizer Semantic/Relational Approach to Science Collaboration</td>
<td>Marriott Salon 12/13</td>
</tr>
<tr>
<td>Tuesday, Sept 23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5:30-6:30 p.m.</td>
<td>Oracle Database 11g Production Case Study: Eli Lilly and Company—Semantic Technologies from a DBA's Point of View</td>
<td>Marriott Salon 01</td>
</tr>
</tbody>
</table>

**DEMOgrounds - Database – Moscone West**

**Oracle Semantic Database Technologies - Pod L14**