What’s New with Oracle Spatial and Graph

Matthew Perry, Ph.D.
Consultant Member of Technical Staff
Oracle Spatial and Graph
September 2018
Safe Harbor Statement

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.
Agenda

1. Introduction to Oracle Spatial and Graph
2. Core Database Features
3. Mid-tier Components and Tools
4. Summary and Thoughts on Standardization
Agenda

1. Introduction to Oracle Spatial and Graph
2. Core Database Features
3. Mid-tier Components and Tools
4. Summary and Thoughts on Standardization
A Little About our Team

• First release of RDF Knowledge Graph was Oracle 10g R2 (2005)

• Active in standards development – team members have been …
  • Member of W3C SPARQL 1.1 WG
  • Member of W3C RDF 1.1 WG
  • Co-editor of W3C OWL 2 Web Ontology Language Profiles spec.
  • Co-editor of W3C R2RML: RDB to RDF Mapping Language spec.
  • Co-editor of OGC GeoSPARQL spec.

• Several research papers in this area
  • VLDB, ICDE, EDBT, ISWC, etc.
Oracle Supports Two Graph Data Models

**Linked Data Semantic Web**
- RDF Knowledge Graph
  - Data Federation
  - Knowledge Representation
  - Semantic Web

**Property Graph**
- Path Analytics
- Social Network Analysis
- Entity Analytics

**Use Case**
- Financial
- Retail, Marketing
- Social Media
- Smart Manufacturing

**Graph Model**
- Life Sciences
- Health Care
- Publishing
- Finance

**Industry Domain**
RDF Knowledge Graph is a Feature of Oracle Database

• Oracle Database is really a multi-model database
  – Relational, Object, Document (XML, JSON), Spatial, Text, Sharded Key-Value, ...
  – RDF Graph – 10gR2
  – Property Graph – 12cR2

• What Database version and options are needed for RDF Knowledge Graph?
  – On-premise
    • Oracle Database Enterprise Edition with Spatial & Graph and Partitioning options
  – Cloud
    • Oracle Database Cloud Service – High Performance or Extreme Performance
    • Oracle Database Exadata Cloud Service
## Oracle Spatial and Graph 18c – RDF Knowledge Graph Features

<table>
<thead>
<tr>
<th>Load / Storage</th>
<th>Query</th>
<th>Reasoning</th>
<th>Analytics</th>
</tr>
</thead>
</table>
| • Fast bulk-load and indexing  
• RDF view of Relational data  
• Manages over a trillion quads | • SPARQL-Jena/Fuseki  
• SPARQL-in-SQL query & update  
• Federated query  
• GeoSPARQL | • RDFS, OWL2 RL, EL, SKOS  
• Ladder-based inference  
• Incremental, parallel reasoning  
• User-defined rules  
• Plug-in architecture | • OBIEE  
• Oracle Advanced Analytics  
• PGX |

### RDF Knowledge Graph

Leverages Oracle capabilities:
- RAC & Exadata scalability
- Compression & partitioning
- In-Memory Column Store
- SQL*Loader direct path load
- Parallel DDL, DML, and query
- RMAN Backup and Recovery
- High Availability
- Oracle Label security
- Enterprise Manager
Oracle Spatial and Graph 18c – RDF Knowledge Graph Architecture

- Protégé Plugin
- Fuseki Endpoint
- Cytoscape Plugin
- SQL Developer RDF Support
- Enterprise Manager and Other DB Tools

Support for Apache Jena (Java API)

Oracle Spatial and Graph 18c features include:

- Generic Relational Schema for Storing RDF Data
- RDF Views of Relational Data
- SPARQL-to-SQL Query Translator
- SPARQL Update Processor
- Forward-chaining OWL Reasoner
- RDF Bulk Loader
- SQL and PL/SQL API
- Support for Apache Jena (Java API)
Agenda

1 Introduction to Oracle Spatial and Graph
2 Core Database Features
3 Mid-tier Components and Tools
4 Summary and Thoughts on Standardization
RDF Bulk Loader

Client

Oracle Protégé Plugin

Jena Adapter Java API

DB Server

Step 1

Step 2

Parallel RDF Bulk Load

- Value Checking
- Canonicalization
- Duplicate Elimination
- Internal ID Generation
- Index Creation

Staging Table

External Table

N-Triple

N-Quad

RDF Relational Schema

N-Triple

N-Quad

Turtle

RDF/XML

Trig

JSON-LD

Others

Copyright © 2018, Oracle and/or its affiliates. All rights reserved.
Core Inference Features in Oracle Database

- Oracle provides native inference in the database for
  - RDFS, RDFS++
  - OWLPRIME, OWL2RL, OWL2EL, SKOSCORE
  - User-defined rules
  - User-defined inferencing

- Inference done using forward chaining
  - Triples inferred and stored ahead of query time
  - Removes on-the-fly reasoning and results in fast query times

- Proof generation
  - Shows one deduction path
Native Inference Engine in Oracle: APIs

**SEM_APIs.CREATE_ENTAILMENT**(
- entailment_name
- sem_models('GraphTBox', 'GraphABox', ...),
- sem_rulebases('OWL2RL'),
- passes,
- inf_components,
- Options,...
)

Use "PROOF=T" to generate inference proof

**SEM_APIs.VALIDATE_ENTAILMENT**(
- sem_models(('GraphTBox', 'GraphABox', ...),
- sem_rulebases('OWL2RL'),
- Criteria,
- Max_conflicts,
- Options
)

**Typical Usage:**
- First load RDF/OWL data
- Call create_entailment to generate inferred graph
- Query both original graph and inferred data
Inferred graph contains only new triples! Saves time & resources

Java API: `performInference`, `deleteInference`, `setInferenceOption`, `analyze` methods in
- `GraphOracleSem`, `DatasetGraphOracleSem` (Jena Adapter)
Entailment with Named Graphs

• Two options
  – Named Graph Local Inference (NGLI)
  – Named Graph Global Inference (NGGI)

<table>
<thead>
<tr>
<th>Asserted Data</th>
</tr>
</thead>
<tbody>
<tr>
<td># TBox:</td>
</tr>
<tr>
<td><a href="">urn:hasSSN</a> rdf:type owl:InverseFunctionalProperty .</td>
</tr>
<tr>
<td><a href="">urn:hasSSN</a> rdfs:domain <a href="">urn:Person</a> .</td>
</tr>
<tr>
<td># ABox:</td>
</tr>
<tr>
<td>GRAPH <a href="">urn:g1</a> {</td>
</tr>
<tr>
<td><a href="">urn:John</a> <a href="">urn:hasSSN</a> 1234 .</td>
</tr>
<tr>
<td>}</td>
</tr>
<tr>
<td>GRAPH <a href="">urn:g2</a> {</td>
</tr>
<tr>
<td><a href="">urn:Johnny</a> <a href="">urn:hasSSN</a> 1234 .</td>
</tr>
<tr>
<td>}</td>
</tr>
</tbody>
</table>

NGLI Entailment

GRAPH <urn:g1> {
  <urn:John> <urn:hasSSN> 1234 .
  <urn:John> rdf:type <urn:Person> .
}
GRAPH <urn:g2> {
  <urn:Johnny> <urn:hasSSN> 1234 .
  <urn:Johnny> rdf:type <urn:Person> .
}

Entails each named graph separately

NGGI Entailment

GRAPH <urn:g1> {
  <urn:John> <urn:hasSSN> 1234 .
}
GRAPH <urn:g2> {
  <urn:Johnny> <urn:hasSSN> 1234 .
}
<urn:Johnny> rdf:type <urn:Person> .
<urn:John> rdf:type <urn:Person> .
<urn:John> owl:sameAs <urn:Johnny> .

Entails union of all named graphs
Enabling Advanced Inference Capabilities

• Parallel inference option
  
  ```
  EXECUTE sem_apis.create_entailment('M_IDX',sem_models('M'),
  sem_rulebases('OWL2RL'), null, null, 'DOP=x');
  ```
  
  Where ‘x’ is the degree of parallelism (DOP)

• Incremental inference option
  
  ```
  EXECUTE sem_apis.create_entailment ('M_IDX',sem_models('M'),
  sem_rulebases('OWL2RL'),null,null, 'INC=T');
  ```

• Enabling owl:sameAs option to limit duplicates
  
  ```
  EXECUTE Sem_apis.create_entailment('M_IDX',sem_models('M'),
  sem_rulebases('OWL2RL') ,null,null, 'OPTSAMEAS=T');
  ```

• Control of individual inference rules used
  
  ```
  EXECUTE Sem_apis.create_entailment('M_IDX',sem_models('M'),
  sem_rulebases('OWL2RL') ,null,'SAM-,SCOH-',null,null);
  ```
  
  For example, disable sameAs generation for individuals and disable subClassOf hierarchy
Extending Semantics Supported by Native OWL Inference Engine

• Option 1: Add user-defined rules
  • Oracle supports, from release 10g, user-defined rules:

<table>
<thead>
<tr>
<th>Antecedents</th>
<th>Consequents</th>
</tr>
</thead>
<tbody>
<tr>
<td>?z :parentOf ?x .</td>
<td></td>
</tr>
<tr>
<td>?x owl:differentFrom ?y .</td>
<td>?x :siblingOf ?y</td>
</tr>
</tbody>
</table>

• Option 2: Leverage external DL reasoners

• Option 3: User-defined inferencing in Oracle Database Release 12c
Extensible Architecture for External OWL DL Reasoners

- External In-Memory OWL DL Reasoners
  - TrOWL/REL

- Jena APIs through Jena Adapter

- Oracle’s Native Inference Engine for OWL 2 RL, EL & user-defined rules

Materialized Inference
Best Practice for Small TBox + Small ABox

• Performing materialisation of the entire ontology with TrOWL
  – Saving full materialisation back to Oracle Spatial and Graph
• No further inference needed in SPARQL querying
  – Because ontology is fully materialised

• Suitable for ontology that can be materialised in memory
• Most efficient and complete SPARQL querying results
Best Practice for Small TBox + Large ABox

• Performing TBox classification only with TrOWL
  – Saving only concept and role hierarchy into Oracle Spatial and Graph

• SPARQL querying powered by Oracle built-in inference engine
  – Further materialize ABox based on the TBox classification computed by TrOWL

• Suitable for ontology with ABox too large to materialize in memory

• Most scalable way handle large ontologies
User-Defined Inferencing in Oracle Database Release 12c

• User-defined inferencing provides more flexibility and control to users
  – Functions including string manipulation, calculations, etc.
    – $?x :firstName  ?fn
    – $?x :lastName   ?ln ⇒ $?x :fullName concatenate(?fn ?ln)
  – Temporal reasoning
    – Overlap of time intervals
  – Geospatial reasoning
  – Calling web Service, …
User-Defined Inferencing in Oracle Database Release 12c

• SDO_SEM_INFERENCE.
  INF_EXT_ACTION_START

• SDO_SEM_INFERENCE.
  INF_EXT_ACTION_RUN

• SDO_SEM_INFERENCE.
  INF_EXT_ACTION_END

```sql
create or replace function sem_inf_example(
  sro_tab_view in varchar2,
  resource_id_map_view in varchar2,
  output_tab in varchar2,
  action in varchar2,
  num_calls in number,
  tппlInferecedLastRound in number,
  options in varchar2 default null,
  optimization_flag out number,
  diag_message out varchar2
) return boolean
as 
pragma autonomous_transaction;
begin
  if (action = SDO_SEM_INFERENCE.INF_EXT_ACTION_START) then
    <!-- preparation work -->
  end if;
  if (action = SDO_SEM_INFERENCE.INF_EXT_ACTION_RUN) then
    <!-- actual inference logic -->
    commit;
  end if;
  if (action = SDO_SEM_INFERENCE.INF_EXT_ACTION_END) then
    <!-- clean up -->
  end if;
return true; -- succeed
end;
/ 
grant execute on sem_inf_example to MDSYS;
```
User-Defined Inferencing in Oracle Database Release 12c

- SDO_SEM_INFERENCE.
  INF_EXT_ACTION_START

- SDO_SEM_INFERENCE.
  INF_EXT_ACTION_RUN

- SDO_SEM_INFERENCE.
  INF_EXT_ACTION_END

http://docs.oracle.com/cd/E16655_01/appdev.121/e17895/inference_extension.htm
SPARQL and **SPARQL in SQL** Architecture

- **HTTP**
  - Standard SPARQL Endpoint
    - Enhanced with query management control

- **Java**
  - Support for Apache Jena

- **Oracle**
  - DATABASE
    - SEM_MATCH
    - SEM_APIS.UPDATE_MODEL

- **SPARQL-to-SQL Core Logic**
SEM_MATCH: Adding SPARQL to SQL

• Extends SQL with full SPARQL 1.1 query constructs
• Benefits:
  – Integrates graph data with existing enterprise data
  – JOINs with other relational (and object-relational) data
  – Allows SQL constructs/functions
  – DDL Statements: create tables/views
  – Allows use of enterprise SQL development tools
SEM_MATCH: Adding SPARQL to SQL

```sql
SELECT n1, n2
FROM TABLE(SM_MATCH('PREFIX foaf: <http://...>
SELECT ?n1 ?n2
FROM <http://g1>
WHERE {?p foaf:name ?n1
    OPTIONAL {?p foaf:knows ?f .
    ?f foaf:name ?n2 }
    FILTER (REGEX(?n1, "^A")) }
ORDER BY ?n1 ?n2',
SEM_MODELS('M1'),...));
```
### SEM_MATCH: Adding SPARQL to SQL

```sql
SELECT n1, n2
FROM TABLE(
  SEM_MATCH('PREFIX foaf: <http://...>
  SELECT ?n1 ?n2
  FROM <http://g1>
  WHERE {?p foaf:name ?n1
    OPTIONAL {?p foaf:knows ?f .
    ?f foaf:name ?n2}
  } FILTER (REGEX(?n1, "^A"))
  ORDER BY ?n1 ?n2',
  SEM_MODELS('M1'),...));
```

<table>
<thead>
<tr>
<th>n1</th>
<th>n2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alex</td>
<td>Jerry</td>
</tr>
<tr>
<td>Alex</td>
<td>Tom</td>
</tr>
<tr>
<td>Alice</td>
<td>Bill</td>
</tr>
<tr>
<td>Alice</td>
<td>Jill</td>
</tr>
<tr>
<td>Alice</td>
<td>John</td>
</tr>
</tbody>
</table>
SPARQL 1.1 Update

- SPARQL 1.1 Update is intended to be a standard language for specifying and executing updates to RDF graphs in a Graph Store.

<table>
<thead>
<tr>
<th>Graph Update operations</th>
<th>Graph Management operations</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>INSERT DATA</strong> Adds some triples, given inline in the request, into a graph.</td>
<td><strong>CREATE</strong> Creates a new graph in stores that support empty graphs.</td>
</tr>
<tr>
<td><strong>DELETE DATA</strong> Removes some triples, given inline in the request, if the respective graph contains those.</td>
<td><strong>DROP</strong> Removes a graph and all of its contents.</td>
</tr>
<tr>
<td><strong>DELETE/INSERT</strong> Perform pattern-based actions for graph updates</td>
<td><strong>COPY</strong> Modifies a graph to contain a copy of another.</td>
</tr>
<tr>
<td><strong>LOAD</strong> Reads the contents of a document representing a graph into a graph in the Graph Store.</td>
<td><strong>MOVE</strong> Moves all of the data from one graph into another.</td>
</tr>
<tr>
<td><strong>CLEAR</strong> Removes all the triples in (one or more) graphs.</td>
<td><strong>ADD</strong> Reproduces all data from one graph into another.</td>
</tr>
</tbody>
</table>
Top-level Procedure: SEM_APIs.UPDATE_MODEL

```java
begin
  sem_api.update_model(
    apply_model=>'M1',
    update_stmt=>'INSERT {?s :mbox ?n} WHERE {?s :email ?n}
  );
end;
/
```
Relational Data to RDF (W3C RDB2RDF)

Two types of mapping: Direct and R2RML

• **Direct Mapping**
  – Automatically generates a mapping based on an input relational schema

• **R2RML (RDB to RDF Mapping Language)**
  – Language for expressing customized mappings
Using Direct Mapping: Overall Flow

- **Source Relational Database**
  - Tables and views
  - Direct Map Author
  - Schema: Classes and Predicates
  - Map: R2RML mapping
  - Query Writer
  - SPARQL QUERY
  - SPARQL to SQL Translator
  - R2RML Processor
Using R2RML: Overall Flow

- **Source Relational Database**
- **R2RML Document**
- **R2RML Map Author**
- **Query Writer**
  - **Schema:** Classes and Predicates
  - **Map:** R2RML mapping
- **R2RML Processor**
- **SPARQL QUERY**
- **SPARQL to SQL Translator**

- Query Writer uses the **Schema** and **Map** to generate a **SPARQL QUERY** which is then translated to **SQL** by the **SPARQL to SQL Translator**.
Oracle Database as an Integration Platform

Oracle Big Data Connectors

Oracle Database Gateways

SPARQL SERVICE

My SQL

DB2

SQL Server

My SQL

Oracle Database Gateways

OBIEE

Oracle Advanced Analytics

RDF View

Relational Table/View

External Table / DB Link

SPARQL/SQL
Agenda

1. Introduction to Oracle Spatial and Graph
2. Core Database Features
3. Mid-tier Components and Tools
4. Summary and Thoughts on Standardization
Support for Apache Jena
Support for Apache Jena

• Implements Apache Jena APIs
  • Popular Java APIs for semantic web based applications
  • Adds Oracle-specific extensions

• Jena support provides three core features:
  • Java APIs for RDF Semantic Graph
    • Data loading, querying, inferencing
  • SPARQL Endpoint for Oracle with SPARQL 1.1 support
  • Oracle-specific extensions for query execution control and management
Provide Convenient Java API for RDF Semantic Graph

- “Proxy” like design
  - Data not cached in memory for scalability
  - SPARQL query converted into SQL and executed inside DB
  - Inference pushed down to the database for execution

- Various data loading methods
  - Bulk/Incremental load RDF or OWL (stored in FS/HDFS/Website) in a variety of formats with strict syntax verification and long literal support
    - N3
    - Turtle
    - RDF/XML
    - N-TRIPLE
    - N-Quads
    - TriG
Support for Protégé 5
Oracle Spatial and Graph Support for Protégé 5

• Two key goals
  – Integrate all the powerful ontology editing capabilities of Protégé
  – Use the same GUI as a unified interface to manage semantic data stored in Oracle Database
Architecture of RDF Semantic Graph Support for Protégé

Oracle Database

OWL API Mapping to RDF

Triples

Write

Read

protégé
Key Features for Protégé Support

• Incremental Edit Saving
• Advanced Bulk Loading
• XML Catalog – Ontology Libraries
  – Support owl:import of ontologies stored in Oracle Database
• Models Manager
• Entailments Manager
• SPARQL Query Worksheet
• SQL Query Worksheet
Oracle SQL Developer RDF Support
Oracle SQL Developer RDF Support

• Since SQL Dev. Version 18.1
• Primary Goals
  – Serve as the primary GUI for RDF Knowledge Graph SQL and PL/SQL APIs
  – Single GUI for both Knowledge Graph and general database operations
  – Provide a familiar interface to Oracle Database users
  – Provide a friendly interface for Semantic Web experts
Oracle SQL Developer RDF Support

• Expand RDF Semantic Graph to see various menu items
  – Models, Rulebases, Entailments, ...
• Right-Click for operations
• Forms to build up argument lists for PL/SQL APIs
Oracle SQL Developer RDF Support

• SPARQL Editor
  – Opens after selecting a model, virtual model, or RDF view
  – Provides a pure-SPARQL query interface
  – Several query templates are available (e.g., distinct predicates)
  – Syntax highlighting and auto-completion
Oracle SQL Developer RDF Support

• R2RML Mapping Editor
  – Shows a tree view of an R2RML mapping
  – Allows drill down from Triples Map -> Subject Map -> Predicate-Object Map
  – You can edit or delete existing elements or add new ones
  – “Commit Mapping” refreshes the associated RDF view
Agenda

1. Introduction to Oracle Spatial and Graph
2. Core Database Features
3. Mid-tier Components and Tools
4. Summary and Thoughts on Standardization
Oracle Spatial and Graph 18c – RDF Knowledge Graph Features

### RDF Knowledge Graph

**Leverages Oracle capabilities:**
- RAC & Exadata scalability
- Compression & partitioning
- In-Memory Column Store
- SQL*Loader direct path load
- Parallel DDL, DML, and query
- RMAN Backup and Recovery
- High Availability
- Oracle Label security
- Enterprise Manager

<table>
<thead>
<tr>
<th>Load / Storage</th>
<th>Query</th>
<th>Reasoning</th>
<th>Analytics</th>
</tr>
</thead>
</table>
| • Fast bulk-load and indexing  
  • RDF view of Relational data  
  • Manages over a trillion quads | • SPARQL-Jena/Fuseki  
  • SPARQL-in-SQL query & update  
  • Federated query  
  • GeoSPARQL | • RDFS, OWL2 RL, EL, SKOS  
  • Ladder-based inference  
  • Incremental, parallel reasoning  
  • User-defined rules  
  • Plug-in architecture | • OBIEE  
• Oracle Advanced Analytics  
• PGX |
Thoughts on Standardization

• Influence from Property Graph community
  – Edge Properties
    • Alternatives for RDF Reification
      – Overloaded named graph – each statement has distinct graph id\(^1\)
      – RDF*/SPARQL* proposal
  – Query Language
    • Possible Extensions
      – Need for better path query support in SPARQL
      – Clear distinction between edge and property – SPARQL only has triple patterns (can SHACL help?)
    • Motivated by
      – Neo4J Cypher, Apache Gremlin, Oracle PGQL, INCITS/DM32.2 SQL-PG

Thoughts on Standardization

• Steep learning curve with Ontologies / OWL
  – Hard for non-experts to understand
    • “The people who maintain these systems will never understand how this works.”
  – There is a need for simple, user-defined rules (e.g., age $\geq 18 \rightarrow$ adult)
    • RIF, SPIN, SWRL

• Make it easy to get started
  – Modularity - example OGC SSN Ontology
    • SOSA – broader audience, allows Schema.org style content enrichment
    • SSN – layers on complexity and expressiveness to SOSA with additional axioms
  – Clearly document which OWL Profile is required
Integrated Cloud
Applications & Platform Services
Native Inference Engine in Oracle

- Implementing an Inference Engine for RDFS/OWL Constructs, ICDE 2008
- Optimizing Enterprise-scale OWL 2 RL Reasoning in a Relational Database System, ISWC 2010
- Advancing the Enterprise-class OWL Inference Engine in Oracle Database, ORE 2012
- Making the Most of your Triple Store: Query Answering in OWL 2 Using an RL Reasoner, WWW 2013

• Leverage SQL and relational technologies (partitioning, compression)
  e.g. RDFS9 Rule Implemented in SQL

```sql
select distinct T2.SID,
  ID(rdf:type),
  T1.OID
from <IVIEW> T1, <IVIEW> T2
where T1.PID=ID(rdfs:subClassOf)
  and T2.PID=ID(rdf:type)
  and T1.SID=T2.OID
  and not exists (
    select 1 from <IVIEW> m
    where m.SID=T2.SID
    and m.PID=ID(rdf:type)
    and m.OID=T1.OID)
```

Parallel Execution
18c New Feature: Composite Partitioning

- Semantic networks can be created with list-hash composite partitioning
- RDF Model – named container of quads with access control
- RDF quads are list partitioned by model id and then subpartitioned by a hash of the predicate id
- Improved query performance through increased parallelism and better optimizer statistics
18c New Feature: In-Memory Virtual Column Support

Materialized Virtual Columns

<table>
<thead>
<tr>
<th>subj_id</th>
<th>subj_lexval</th>
<th>pred_id</th>
<th>pred_lexval</th>
<th>obj_id</th>
<th>obj_lexval</th>
<th>graph_id</th>
<th>graph_lexval</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td><a href="">urn:abc</a></td>
<td>456</td>
<td><a href="">urn:pred1</a></td>
<td>789</td>
<td>&quot;def&quot;</td>
<td>102</td>
<td><a href="">urn:g1</a></td>
</tr>
<tr>
<td>100</td>
<td><a href="">urn:uvw</a></td>
<td>456</td>
<td><a href="">urn:pred1</a></td>
<td>101</td>
<td>&quot;xyz&quot;</td>
<td>102</td>
<td><a href="">urn:g1</a></td>
</tr>
</tbody>
</table>

In Memory (columnar)

<table>
<thead>
<tr>
<th>value_id</th>
<th>lex_val</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td><a href="">urn:abc</a></td>
</tr>
<tr>
<td>456</td>
<td><a href="">urn:pred1</a></td>
</tr>
<tr>
<td>789</td>
<td>&quot;def&quot;</td>
</tr>
<tr>
<td>100</td>
<td><a href="">urn:uvw</a></td>
</tr>
<tr>
<td>101</td>
<td>&quot;xyz&quot;</td>
</tr>
<tr>
<td>102</td>
<td><a href="">urn:g1</a></td>
</tr>
</tbody>
</table>

On Disk

<table>
<thead>
<tr>
<th>subj_id</th>
<th>pred_id</th>
<th>obj_id</th>
<th>graph_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>456</td>
<td>789</td>
<td>102</td>
</tr>
<tr>
<td>100</td>
<td>456</td>
<td>101</td>
<td>102</td>
</tr>
</tbody>
</table>

value_id lex_val
123 <urn:abc>
456 <urn:pred1>
789 "def"
100 <urn:uvw>
101 "xyz"
102 <urn:g1>
Jena Support As Integration Bridge

• Allows **integration** of RDF Semantic Graph with various tools
  • TopQuadrant Tools
    • TopBraid Composer
    • TopBraid Live (intelligent business applications platform)
    • TopBraid Insight (on demand integrated access to diverse data)
  • OBIEE
    • SPARQL Gateway defines a physical data source to OBI
  • External OWL DL reasoners
    • TrOWL/REL
  • Open-source ontology editing tool
    • Protégé
  • Hadoop ecosystem/Big Data Appliance (BDA)
    • Loader in Jena Support can easily read from HDFS
SPARQL Endpoint with Joseki/Fuseki

• SPARQL service endpoint supporting full **SPARQL Protocol**
  – Integrated with Jena/Joseki 3.4.4 /Fuseki 2.4.0 (deployed in WLS, Tomcat, or JBoss)
  – Joseki uses J2EE data source for DB connection specification
  – SPARQL 1.1 query and update supported

• Oracle-specific declarative configuration options in Joseki
  – Each URI endpoint is mapped to a Joseki service:

```xml
<#service>
  rdf:type                joseki:Service ;
  rdfs:label             "SPARQL with Oracle Semantic Data Management" ;
  joseki:serviceRef     "oracle" ; #web.xml must route this name to Joseki
  joseki:dataset        <#oracle> ; # dataset part
  joseki:processor      joseki:ProcessorSPARQL_FixedDS;
</#service>
```
Key Features for Protégé Support

• Seamless integration with Oracle Database
  – Uses Jena Adapter to communicate with Oracle Database
  – Create and modify models
  – Graphic interface to interact with models
• Incremental Edit Saving
• Advanced Bulk Loading
• OWL Parser
  – Web Ontology Language Mapping to RDF Graphs
  – Strict implementation of W3C Recommendations
Key Features for Protégé Support

- The support provides clear and easy to use tools to connect and interact seamlessly with an Oracle Database.
- Advanced Security is provided by Oracle Label Security, which provides Triple Level Security and transparent integration on top of Protégé.