Put SPARQL in Your Code: Building Applications with Oracle Semantic Technologies

Xavier Lopez, Ph.D.
Zhe Wu, Ph.D.
Sourpriya Das, Ph.D.
<table>
<thead>
<tr>
<th>Date/Time</th>
<th>Title</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sunday, Oct. 11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10:30 a.m.</td>
<td>Build Better SQL and Document Queries with Oracle</td>
<td>Hilton Hotel Continental Parlor 1/2/3</td>
</tr>
<tr>
<td></td>
<td>Semantic Technologies, Part 1</td>
<td></td>
</tr>
<tr>
<td>11:45 a.m.</td>
<td>Build Better SQL and Document Queries with Oracle</td>
<td>Hilton Hotel Continental Parlor 1/2/3</td>
</tr>
<tr>
<td></td>
<td>Semantic Technologies, Part 2</td>
<td></td>
</tr>
<tr>
<td>2:30 p.m.</td>
<td>Dive into Semantic Technologies: Security, Document Index,</td>
<td>Hilton Hotel Golden Gate 2</td>
</tr>
<tr>
<td></td>
<td>and Change Management</td>
<td></td>
</tr>
<tr>
<td>3:45 p.m.</td>
<td>Put SPARQL in Your Code: Building Applications with Oracle</td>
<td>Hilton Hotel Golden Gate 2</td>
</tr>
<tr>
<td></td>
<td>Semantic Technologies</td>
<td></td>
</tr>
<tr>
<td>Tuesday, Oct. 13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4:00 p.m.</td>
<td>Turning the Database Inside-Out with Complex Event Processing</td>
<td>Hilton Hotel Golden Gate 1</td>
</tr>
</tbody>
</table>
### Semantics at OOW 2009 - Sessions

<table>
<thead>
<tr>
<th>Date/Time</th>
<th>Title</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wednesday, Oct. 14</td>
<td>Who’s Who and What’s What with Oracle Database Semantic Technologies</td>
<td>Moscone South Room 270</td>
</tr>
</tbody>
</table>

- DEMOgrounds
  - Oracle Database Semantic Technologies - *Moscone West, W-018*
Agenda

• Intro to Semantic Web and Business Use Cases
• Semantic Technologies in Oracle Database 11gR2
  • Fundamentals: Storage and Query
  • Enterprise RDF: Semantic indexing, Security, Versioning
  • Inference
• Jena Adaptor for Oracle Database
• Utilities
• Performance
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 DECISION. THE DEVELOPMENT, RELEASE, AND TIMING OF ANY FEATURES OR FUNCTIONALITY DESCRIBED FOR ORACLE'S PRODUCTS REMAINS AT THE SOLE DISCRETION OF ORACLE.
Introduction to Semantic Web and Business Use Cases
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
Case Study: National Intelligence

Information Extraction
- Categorization, Feature/term Extraction

Processed Document Collection
- RDF/OWL

Ontology Engineering Modeling Process

Domain Specific Knowledge Base
- OWL Ontologies

SQL/SPARQL Query

Explore

Analyst

Browsing, Presentation, Reporting, Visualization, Query

Web Resources
- News, Email, RSS

Content Mgmt. Systems

Processed Document Collection

Ontology Engineering Modeling Process
Data Integration Platform in Health Informatics

Enterprise Information Consumers (EICs)

- Patient Care
- Workforce Management
- Business Intelligence
- Clinical Analytics

Integration Server (Semantic Knowledge base)

Run-Time Metadata

Access

Deploy

Model Virtual

Relate

Model Physical

LIS  CIS  HTB  HIS
Data Integration Platform in Health Informatics

Enterprise Information Consumers (EICs)

Patient Care
Workforce Management
Business Intelligence
Clinical Analytics

Design-Time Metadata

Integration Server (Semantic Knowledge base)

Run-Time Metadata

Access

Deploy

Model Virtual

Relate

Model Physical

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

Partners

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

Partners
Oracle database 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 (billions of triples)
- Choice of SQL or SPARQL query
- Leverages Oracle Partitioning. 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
  - OWLprime, OWL 2 RL, RDFS
  - User defined rule base
Semantic Technology Partners
Integrated Tools and Solution Providers:
Release 11.1 Semantic Technologies

- W3C standards: RDF, RDFS, OWL, SPARQL
- SQL & DML access to RDF/OWL data and ontologies
  - Querying in SQL w/ SPARQL-like graph patterns
  - Querying w/ SPARQL in Java through Jena adaptor
  - Ontology-assisted querying of relational data
- Native Inferencing: RDFS/OWL & user-defined rules
  - Persistent, scalable, ahead of queries, pluggable reasoners
- Model level security
- Bulk and incremental loading of triples
  - Up to 8 exabytes, 1 million graphs w/ multi-graph querying
  - Integral Oracle Partitioning and also table compression
Release 11.2 Semantic Technologies

• Strong security for Semantic Technologies
  – Security policies and data classification for RDF data

• Semantic indexing of documents
  – Semantic indexing of documents based on popular natural language tools

• Faster, more efficient reasoning to find new relationships
  – Parallel and incremental inference, owl:sameAs optimization

• Change management for collaboration

• Standards & open source support
  – SPARQL query support for Filter, Union in SEM_MATCH table function
  – OWL: union, intersection, OWL 2 property chains, disjoint properties
  – Pellet OWL DL reasoner Integration
  – Jena v2.6, ARQ 2.7.0
  – Java SDK for SPARQL for 3rd party integration e.g., Sesame
  – W3C Simple Knowledge Organization System (SKOS) & SNOMED ontology
  – Utility to swap, rename, and merge model, rename entailment, remove duplicates
Background
Table of Family data: converting to RDF

<table>
<thead>
<tr>
<th>Name</th>
<th>hasFather</th>
<th>hasMother</th>
<th>hasSister</th>
</tr>
</thead>
<tbody>
<tr>
<td>sn:Cathy</td>
<td>sn:Sammy</td>
<td>sn:Suzie</td>
<td></td>
</tr>
<tr>
<td>sn:Jack</td>
<td>sn:Sammy</td>
<td>sn:Suzie</td>
<td>sn:Cathy</td>
</tr>
<tr>
<td>sn:Tom</td>
<td>sn:Matt</td>
<td>sn:Martha</td>
<td></td>
</tr>
<tr>
<td>sn:Cindy</td>
<td>sn:Matt</td>
<td>sn:Martha</td>
<td></td>
</tr>
<tr>
<td>sn:Suzie</td>
<td>sn:John</td>
<td>sn:Janice</td>
<td></td>
</tr>
<tr>
<td>sn:Matt</td>
<td>sn:John</td>
<td>sn:Janice</td>
<td></td>
</tr>
</tbody>
</table>

sn = http:/www.socialnetwork.com
### Table of Family data: converting to RDF

<table>
<thead>
<tr>
<th>Name</th>
<th>hasFather</th>
<th>hasMother</th>
<th>hasSister</th>
</tr>
</thead>
<tbody>
<tr>
<td>sn:Cathy</td>
<td>sn:Sammy</td>
<td>sn:Suzie</td>
<td></td>
</tr>
<tr>
<td>sn:Jack</td>
<td>sn:Sammy</td>
<td>sn:Suzie</td>
<td>sn:Cathy</td>
</tr>
<tr>
<td>sn:Tom</td>
<td>sn:Matt</td>
<td>sn:Martha</td>
<td></td>
</tr>
<tr>
<td>sn:Cindy</td>
<td>sn:Matt</td>
<td>sn:Martha</td>
<td></td>
</tr>
<tr>
<td>sn:Suzie</td>
<td>sn:John</td>
<td>sn:Janice</td>
<td></td>
</tr>
<tr>
<td>sn:Matt</td>
<td>sn:John</td>
<td>sn:Janice</td>
<td></td>
</tr>
</tbody>
</table>

sn = http://www.socialnetwork.com
## Table of Family data: converting to RDF

<table>
<thead>
<tr>
<th>Name</th>
<th>hasFather</th>
<th>hasMother</th>
<th>hasSister</th>
</tr>
</thead>
<tbody>
<tr>
<td>sn:Cathy</td>
<td>sn:Sammy</td>
<td>sn:Suzie</td>
<td></td>
</tr>
<tr>
<td>sn:Jack</td>
<td>sn:Sammy</td>
<td>sn:Suzie</td>
<td>sn:Cathy</td>
</tr>
<tr>
<td>sn:Tom</td>
<td>sn:Matt</td>
<td>sn:Martha</td>
<td></td>
</tr>
<tr>
<td>sn:Cindy</td>
<td>sn:Matt</td>
<td>sn:Martha</td>
<td></td>
</tr>
<tr>
<td>sn:Suzie</td>
<td>sn:John</td>
<td>sn:Janice</td>
<td></td>
</tr>
<tr>
<td>sn:Matt</td>
<td>sn:John</td>
<td>sn:Janice</td>
<td></td>
</tr>
</tbody>
</table>
### Table of Family data: converting to RDF

<table>
<thead>
<tr>
<th>Name</th>
<th>hasFather</th>
<th>hasMother</th>
<th>hasSister</th>
</tr>
</thead>
<tbody>
<tr>
<td>sn:Cathy</td>
<td>sn:Sammy</td>
<td>sn:Suzie</td>
<td></td>
</tr>
<tr>
<td>sn:Jack</td>
<td>sn:Sammy</td>
<td>sn:Suzie</td>
<td>sn:Cathy</td>
</tr>
<tr>
<td>sn:Tom</td>
<td>sn:Matt</td>
<td>sn:Martha</td>
<td></td>
</tr>
<tr>
<td>sn:Cindy</td>
<td>sn:Matt</td>
<td>sn:Martha</td>
<td></td>
</tr>
<tr>
<td>sn:Suzie</td>
<td>sn:John</td>
<td>sn:Janice</td>
<td></td>
</tr>
<tr>
<td>sn:Matt</td>
<td>sn:John</td>
<td>sn:Janice</td>
<td></td>
</tr>
</tbody>
</table>

sn = http://www.socialnetwork.com
<table>
<thead>
<tr>
<th>Name</th>
<th>hasFather</th>
<th>hasMother</th>
<th>hasSister</th>
</tr>
</thead>
<tbody>
<tr>
<td>sn:Cathy</td>
<td>sn:Sammy</td>
<td>sn:Suzie</td>
<td></td>
</tr>
<tr>
<td>sn:Jack</td>
<td>sn:Sammy</td>
<td>sn:Suzie</td>
<td>sn:Cathy</td>
</tr>
<tr>
<td>sn:Tom</td>
<td>sn:Matt</td>
<td>sn:Martha</td>
<td></td>
</tr>
<tr>
<td>sn:Cindy</td>
<td>sn:Matt</td>
<td>sn:Martha</td>
<td></td>
</tr>
<tr>
<td>sn:Suzie</td>
<td>sn:John</td>
<td>sn:Janice</td>
<td></td>
</tr>
<tr>
<td>sn:Matt</td>
<td>sn:John</td>
<td>sn:Janice</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>sn:Cathy</td>
<td>hasFather</td>
<td>sn:Sammy</td>
</tr>
<tr>
<td>sn:Cathy</td>
<td>hasMother</td>
<td>sn:Suzie</td>
</tr>
<tr>
<td>sn:Jack</td>
<td>hasFather</td>
<td>sn:Sammy</td>
</tr>
<tr>
<td>sn:Jack</td>
<td>hasMother</td>
<td>sn:Suzie</td>
</tr>
<tr>
<td>sn:Jack</td>
<td>hasSister</td>
<td>sn:Cathy</td>
</tr>
</tbody>
</table>
## Table of Family data: converting to RDF

<table>
<thead>
<tr>
<th>Name</th>
<th>hasFather</th>
<th>hasMother</th>
<th>hasSister</th>
</tr>
</thead>
<tbody>
<tr>
<td>sn:Cathy</td>
<td>sn:Sammy</td>
<td>sn:Suzie</td>
<td></td>
</tr>
<tr>
<td>sn:Jack</td>
<td>sn:Sammy</td>
<td>sn:Suzie</td>
<td>sn:Cathy</td>
</tr>
<tr>
<td>sn:Tom</td>
<td>sn:Matt</td>
<td>sn:Martha</td>
<td></td>
</tr>
<tr>
<td>sn:Cindy</td>
<td>sn:Matt</td>
<td>sn:Martha</td>
<td></td>
</tr>
<tr>
<td>sn:Suzie</td>
<td>sn:John</td>
<td>sn:Janice</td>
<td></td>
</tr>
<tr>
<td>sn:Matt</td>
<td>sn:John</td>
<td>sn:Janice</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>sn:Cathy</td>
<td>hasFather</td>
<td>sn:Sammy</td>
</tr>
<tr>
<td>sn:Cathy</td>
<td>hasMother</td>
<td>sn:Suzie</td>
</tr>
<tr>
<td>sn:Jack</td>
<td>hasFather</td>
<td>sn:Sammy</td>
</tr>
<tr>
<td>sn:Jack</td>
<td>hasMother</td>
<td>sn:Suzie</td>
</tr>
<tr>
<td>sn:Jack</td>
<td>hasSister</td>
<td>sn:Cathy</td>
</tr>
</tbody>
</table>
Table of Family data: converting to RDF

<table>
<thead>
<tr>
<th>Name</th>
<th>hasFather</th>
<th>hasMother</th>
<th>hasSister</th>
</tr>
</thead>
<tbody>
<tr>
<td>sn:Cathy</td>
<td>sn:Sammy</td>
<td>sn:Suzie</td>
<td></td>
</tr>
<tr>
<td>sn:Jack</td>
<td>sn:Sammy</td>
<td>sn:Suzie</td>
<td>sn:Cathy</td>
</tr>
<tr>
<td>sn:Tom</td>
<td></td>
<td></td>
<td>sn:Martha</td>
</tr>
<tr>
<td>sn:Cindy</td>
<td>sn:Matt</td>
<td>sn:Martha</td>
<td></td>
</tr>
<tr>
<td>sn:Suzie</td>
<td>sn:John</td>
<td>sn:Janice</td>
<td></td>
</tr>
<tr>
<td>sn:Matt</td>
<td>sn:John</td>
<td>sn:Janice</td>
<td></td>
</tr>
</tbody>
</table>

Subject | Predicate  | Object   |
-------|------------|----------|
sn:Cathy | hasFather  | sn:Sammy|
sn:Cathy | hasMother  | sn:Suzie|
sn:Jack  | hasFather  | sn:Sammy|
sn:Jack  | hasMother  | sn:Suzie|
sn:Jack  | hasSister  | sn:Cathy|
### Table of Family data: inferring new data

<table>
<thead>
<tr>
<th>Name</th>
<th>hasFather</th>
<th>hasMother</th>
<th>hasSister</th>
</tr>
</thead>
<tbody>
<tr>
<td>sn:Cathy</td>
<td>sn:Sammy</td>
<td>sn:Suzie</td>
<td></td>
</tr>
<tr>
<td>sn:Jack</td>
<td>sn:Sammy</td>
<td>sn:Suzie</td>
<td>sn:Cathy</td>
</tr>
<tr>
<td>sn:Tom</td>
<td>sn:Matt</td>
<td>sn:Martha</td>
<td></td>
</tr>
<tr>
<td>sn:Cindy</td>
<td>sn:Matt</td>
<td>sn:Martha</td>
<td></td>
</tr>
<tr>
<td>sn:Suzie</td>
<td>sn:John</td>
<td>sn:Janice</td>
<td></td>
</tr>
<tr>
<td>sn:Matt</td>
<td>sn:John</td>
<td>sn:Janice</td>
<td></td>
</tr>
</tbody>
</table>
# Table of Family data: inferring new data

<table>
<thead>
<tr>
<th>Name</th>
<th>hasFather</th>
<th>hasMother</th>
<th>hasSister</th>
</tr>
</thead>
<tbody>
<tr>
<td>sn:Cathy</td>
<td>sn:Sammy</td>
<td>sn:Suzie</td>
<td></td>
</tr>
<tr>
<td>sn:Jack</td>
<td>sn:Sammy</td>
<td>sn:Suzie</td>
<td>sn:Cathy</td>
</tr>
<tr>
<td>sn:Tom</td>
<td>sn:Matt</td>
<td>sn:Martha</td>
<td></td>
</tr>
<tr>
<td>sn:Cindy</td>
<td>sn:Matt</td>
<td>sn:Martha</td>
<td></td>
</tr>
<tr>
<td>sn:Suzie</td>
<td>sn:John</td>
<td>sn:Janice</td>
<td></td>
</tr>
<tr>
<td>sn:Matt</td>
<td>sn:John</td>
<td>sn:Janice</td>
<td></td>
</tr>
</tbody>
</table>
Table of Family data: inferring new data

<table>
<thead>
<tr>
<th>Name</th>
<th>hasFather</th>
<th>hasMother</th>
<th>hasSister</th>
</tr>
</thead>
<tbody>
<tr>
<td>sn:Cathy</td>
<td>sn:Sammy</td>
<td>sn:Suzie</td>
<td></td>
</tr>
<tr>
<td>sn:Jack</td>
<td>sn:Sammy</td>
<td>sn:Suzie</td>
<td>sn:Cathy</td>
</tr>
<tr>
<td>sn:Tom</td>
<td>sn:Matt</td>
<td>sn:Martha</td>
<td></td>
</tr>
<tr>
<td>sn:Cindy</td>
<td>sn:Matt</td>
<td>sn:Martha</td>
<td></td>
</tr>
<tr>
<td>sn:Suzie</td>
<td>sn:John</td>
<td>sn:Janice</td>
<td></td>
</tr>
<tr>
<td>sn:Matt</td>
<td>sn:John</td>
<td>sn:Janice</td>
<td></td>
</tr>
</tbody>
</table>
## Table of Family data: inferring new data

<table>
<thead>
<tr>
<th>Name</th>
<th>hasFather</th>
<th>hasMother</th>
<th>hasSister</th>
</tr>
</thead>
<tbody>
<tr>
<td>sn:Cathy</td>
<td>sn:Sammy</td>
<td>sn:Suzie</td>
<td></td>
</tr>
<tr>
<td>sn:Jack</td>
<td>sn:Sammy</td>
<td>sn:Suzie</td>
<td>sn:Cathy</td>
</tr>
<tr>
<td>sn:Tom</td>
<td>sn:Matt</td>
<td>sn:Martha</td>
<td></td>
</tr>
<tr>
<td>sn:Cindy</td>
<td>sn:Matt</td>
<td>sn:Martha</td>
<td></td>
</tr>
<tr>
<td>sn:Suzie</td>
<td>sn:John</td>
<td>sn:Janice</td>
<td></td>
</tr>
<tr>
<td>sn:Matt</td>
<td>sn:John</td>
<td>sn:Janice</td>
<td></td>
</tr>
<tr>
<td>sn:Sammy</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sn:Janice</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sn:Martha</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sn:John</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Table of Family data: inferring new data

<table>
<thead>
<tr>
<th>Name</th>
<th>hasFather</th>
<th>hasMother</th>
<th>hasSister</th>
</tr>
</thead>
<tbody>
<tr>
<td>sn:Cathy</td>
<td>sn:Sammy</td>
<td>sn:Suzie</td>
<td></td>
</tr>
<tr>
<td>sn:Jack</td>
<td>sn:Sammy</td>
<td>sn:Suzie</td>
<td>sn:Cathy</td>
</tr>
<tr>
<td>sn:Tom</td>
<td>sn:Matt</td>
<td>sn:Martha</td>
<td></td>
</tr>
<tr>
<td>sn:Cindy</td>
<td>sn:Matt</td>
<td>sn:Martha</td>
<td></td>
</tr>
<tr>
<td>sn:Suzie</td>
<td>sn:John</td>
<td>sn:Janice</td>
<td></td>
</tr>
<tr>
<td>sn:Matt</td>
<td>sn:John</td>
<td>sn:Janice</td>
<td></td>
</tr>
<tr>
<td>sn:Sammy</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sn:Janice</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sn:Martha</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sn:John</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Table of Family data: inferring new data

<table>
<thead>
<tr>
<th>Name</th>
<th>hasFather</th>
<th>hasMother</th>
<th>hasSister</th>
</tr>
</thead>
<tbody>
<tr>
<td>sn:Cathy</td>
<td>sn:Sammy</td>
<td>sn:Suzie</td>
<td></td>
</tr>
<tr>
<td>sn:Jack</td>
<td>sn:Sammy</td>
<td>sn:Suzie</td>
<td>sn:Cathy</td>
</tr>
<tr>
<td>sn:Tom</td>
<td>sn:Matt</td>
<td>sn:Martha</td>
<td></td>
</tr>
<tr>
<td>sn:Cindy</td>
<td>sn:Matt</td>
<td>sn:Martha</td>
<td></td>
</tr>
<tr>
<td>sn:Suzie</td>
<td>sn:John</td>
<td>sn:Janice</td>
<td></td>
</tr>
<tr>
<td>sn:Matt</td>
<td>sn:John</td>
<td>sn:Janice</td>
<td></td>
</tr>
<tr>
<td>sn:Sammy</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sn:Janice</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sn:Martha</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sn:John</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Table of Family data: inferring new data

<table>
<thead>
<tr>
<th>Name</th>
<th>hasFather</th>
<th>hasMother</th>
<th>hasSister</th>
<th>Gender</th>
</tr>
</thead>
<tbody>
<tr>
<td>sn:Cathy</td>
<td>sn:Sammy</td>
<td>sn:Suzie</td>
<td></td>
<td>Female</td>
</tr>
<tr>
<td>sn:Jack</td>
<td>sn:Sammy</td>
<td>sn:Suzie</td>
<td>sn:Cathy</td>
<td></td>
</tr>
<tr>
<td>sn:Tom</td>
<td>sn:Matt</td>
<td>sn:Martha</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sn:Cindy</td>
<td>sn:Matt</td>
<td>sn:Martha</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sn:Suzie</td>
<td>sn:John</td>
<td>sn:Janice</td>
<td></td>
<td>Female</td>
</tr>
<tr>
<td>sn:Matt</td>
<td>sn:John</td>
<td>sn:Janice</td>
<td></td>
<td>Male</td>
</tr>
<tr>
<td>sn:Sammy</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sn:Janice</td>
<td></td>
<td></td>
<td></td>
<td>Female</td>
</tr>
<tr>
<td>sn:Martha</td>
<td></td>
<td></td>
<td></td>
<td>Female</td>
</tr>
<tr>
<td>sn:John</td>
<td></td>
<td></td>
<td></td>
<td>Male</td>
</tr>
</tbody>
</table>
SPARQL Query Language for RDF

John
Janice
Sammy
Suzie
Matt
Martha
Cathy
Jack
Tom
Cindy

hasFather
hasMother
SPARQL Query Language for RDF

Querying RDF data

- Find pairs of siblings (same parents)
- SELECT ?x ?y
  FROM <rdf_graph> WHERE {
    ?x hasFather ?f . ?x hasMother ?m .
    ?y hasFather ?f . ?y hasMother ?m .
    FILTER(?x != ?y)
  }
SPARQL Query Language for RDF

Querying RDF data
- Find pairs of siblings (same parents)
- SELECT ?x ?y
  FROM <rdf_graph> WHERE {
    ?x hasFather ?f . ?x hasMother ?m .
    ?y hasFather ?f . ?y hasMother ?m .
    FILTER( ?x != ?y )
  }

```
John Janice
Sammy Suzie Matt
Cathy Cindy
Jack Tom Martha
```

1 hasFather
1 hasMother
SPARQL Query Language for RDF

Querying RDF data

- Find pairs of siblings (same parents)
- SELECT ?x ?y
  FROM <rdf_graph> WHERE {
  ?x hasFather ?f . ?x hasMother ?m .
  ?y hasFather ?f . ?y hasMother ?m .
  FILTER( ?x != ?y )
}

Diagram:
- John
- Janice
- Sammy
- Suzie
- Matt
- Martha
- Cathy
- Tom
- Cindy

Relationships:
- hasFather
- hasMother
SPARQL Query Language for RDF

Querying RDF data

- Find pairs of siblings (same parents)
- SELECT ?x ?y
  FROM <rdf_graph> WHERE {
    ?x hasFather ?f . ?x hasMother ?m .
    ?y hasFather ?f . ?y hasMother ?m .
    FILTER( ?x != ?y )
  }

Network Diagram:
- John
- Janice
- Sammy
- Suzie
- Matt
- Martha
- Cathy
- Jack
- Tom
- Cindy

Edges:
- hasFather
- hasMother
SPARQL Query Language for RDF

Querying RDF data

• Find pairs of siblings (same parents)
• SELECT ?x ?y
  FROM <rdf_graph> WHERE {
    ?x hasFather ?f . ?x hasMother ?m .
    ?y hasFather ?f . ?y hasMother ?m .
    FILTER( ?x != ?y )
  }

1
2
3
4

hasFather

hasMother

Sammy

Suzie

Matt

Martha

Cathy

Jack

Tom

Cindy

John

Janice
Querying RDF data

- Find pairs of siblings (same parents)
- SELECT ?x ?y
  FROM <rdf_graph> WHERE {
    ?x hasFather ?f . ?x hasMother ?m .
    ?y hasFather ?f . ?y hasMother ?m .
    FILTER(?x != ?y)
  }

Result (bindings)

Sammy
  hasFather
  hasMother

Suzie

Matt

Martha

Cathy

Jack

Tom

Cindy
SPARQL Query Language for RDF

Querying RDF data
- Find pairs of siblings (same parents)
- SELECT ?x ?y
  FROM <rdf_graph> WHERE {
    ?x hasFather ?f . ?x hasMother ?m .
    ?y hasFather ?f . ?y hasMother ?m .
    FILTER( ?x != ?y )
  }

Result (bindings)
?x  ?y
SPARQL Query Language for RDF

Querying RDF data
- Find pairs of siblings (same parents)
- SELECT ?x ?y
  FROM <rdf_graph> WHERE {
  ?x hasFather ?f . ?x hasMother ?m .
  ?y hasFather ?f . ?y hasMother ?m .
  FILTER( ?x != ?y )
}
SPARQL Query Language for RDF

Querying RDF data

• Find pairs of siblings (same parents)
• SELECT ?x ?y
  FROM <rdf_graph> WHERE {
    ?x hasFather ?f . ?x hasMother ?m .
    ?y hasFather ?f . ?y hasMother ?m .
    FILTER( ?x != ?y )
  }

Result (bindings)

<table>
<thead>
<tr>
<th>?x</th>
<th>?y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Suzie</td>
<td>Matt</td>
</tr>
<tr>
<td>Matt</td>
<td>Suzie</td>
</tr>
</tbody>
</table>
SPARQL Query Language for RDF

Querying RDF data
• Find pairs of siblings (same parents)
• SELECT ?x ?y
  FROM <rdf_graph> WHERE {
  ?x hasFather ?f . ?x hasMother ?m .
  ?y hasFather ?f . ?y hasMother ?m .
  FILTER( ?x != ?y )
}

Result (bindings)

<table>
<thead>
<tr>
<th>?x</th>
<th>?y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Suzie</td>
<td>Matt</td>
</tr>
<tr>
<td>Matt</td>
<td>Suzie</td>
</tr>
</tbody>
</table>
SPARQL Query Language for RDF

Querying RDF data
- Find pairs of siblings (same parents)
- SELECT ?x ?y
  FROM <rdf_graph> WHERE {
    ?x hasFather ?f . ?x hasMother ?m .
    ?y hasFather ?f . ?y hasMother ?m .
    FILTER( ?x != ?y )
  }

Result (bindings)
- Suzie  Matt
- Matt  Suzie
- Cathy  Jack
SPARQL Query Language for RDF

Querying RDF data

- Find pairs of siblings (same parents)
- SELECT ?x ?y
  FROM <rdf_graph> WHERE {
    ?x hasFather ?f . ?x hasMother ?m .
    ?y hasFather ?f . ?y hasMother ?m .
    FILTER(?x != ?y) }

Result (bindings)

<table>
<thead>
<tr>
<th>?x</th>
<th>?y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Suzie</td>
<td>Matt</td>
</tr>
<tr>
<td>Matt</td>
<td>Suzie</td>
</tr>
<tr>
<td>Cathy</td>
<td>Jack</td>
</tr>
<tr>
<td>Jack</td>
<td>Cathy</td>
</tr>
</tbody>
</table>
SPARQL Query Language for RDF

Querying RDF data

• Find pairs of siblings (same parents)
• SELECT ?x ?y
  FROM <rdf_graph> WHERE {
    ?x hasFather ?f . ?x hasMother ?m .
    ?y hasFather ?f . ?y hasMother ?m .
    FILTER( ?x != ?y )
  }

Result (bindings)

<table>
<thead>
<tr>
<th>?x</th>
<th>?y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Suzie</td>
<td>Matt</td>
</tr>
<tr>
<td>Matt</td>
<td>Suzie</td>
</tr>
<tr>
<td>Cathy</td>
<td>Jack</td>
</tr>
<tr>
<td>Jack</td>
<td>Cathy</td>
</tr>
</tbody>
</table>
SPARQL Query Language for RDF

Querying RDF data
- Find pairs of siblings (same parents)
- SELECT ?x ?y
  FROM <rdf_graph> WHERE {
    ?x hasFather ?f . ?x hasMother ?m .
    ?y hasFather ?f . ?y hasMother ?m .
    FILTER( ?x != ?y )
  }

Result (bindings)

<table>
<thead>
<tr>
<th>?x</th>
<th>?y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Suzie</td>
<td>Matt</td>
</tr>
<tr>
<td>Matt</td>
<td>Suzie</td>
</tr>
<tr>
<td>Cathy</td>
<td>Jack</td>
</tr>
<tr>
<td>Jack</td>
<td>Cathy</td>
</tr>
<tr>
<td>Tom</td>
<td>Cindy</td>
</tr>
</tbody>
</table>
Querying RDF data

- Find pairs of siblings (same parents)
- SELECT ?x ?y
  FROM <rdf_graph> WHERE {

  ?x hasFather ?f . ?x hasMother ?m .
  ?y hasFather ?f . ?y hasMother ?m .
  FILTER(?x != ?y)
  }

Result (bindings)

<table>
<thead>
<tr>
<th>?x</th>
<th>?y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Suzie</td>
<td>Matt</td>
</tr>
<tr>
<td>Matt</td>
<td>Suzie</td>
</tr>
<tr>
<td>Cathy</td>
<td>Jack</td>
</tr>
<tr>
<td>Jack</td>
<td>Cathy</td>
</tr>
<tr>
<td>Tom</td>
<td>Cindy</td>
</tr>
<tr>
<td>Cindy</td>
<td>Tom</td>
</tr>
</tbody>
</table>
SQL and graph query
SQL and graph query

Querying RDF data

• Find pairs of siblings (same parents)
• SELECT ?x ?y
  FROM <rdf_graph>
  WHERE
  {
    ?x hasFather ?f .
    ?x hasMother ?m .
    ?y hasFather ?f .
    ?y hasMother ?m .
  }
  FILTER( ?x != ?y )
Find pairs of siblings (same parents)

```sql
SELECT ?x ?y
FROM <rdf_graph>
WHERE
{
  ?x hasFather ?f .
  ?x hasMother ?m .
  ?y hasFather ?f .
  ?y hasMother ?m .

  FILTER( ?x != ?y )
}
```

```
rdf_graph g1, rdf_graph g2, rdf_graph g3, rdf_graph g4
```
SQL and graph query

Querying RDF data

- Find pairs of siblings (same parents)
  
  ```sql
  SELECT g1.subject x, g3.subject y
  FROM rdf_graph g1, rdf_graph g2, rdf_graph g3, rdf_graph g4
  WHERE
  {
    ?x hasFather ?f .
    ?x hasMother ?m .
    ?y hasFather ?f .
    ?y hasMother ?m .
  }
  FILTER( ?x != ?y )
  ```
SQL and graph query

Querying RDF data

• Find pairs of siblings (same parents)

```
SELECT g1.subject x, g3.subject y
FROM rdf_graph g1, rdf_graph g2, rdf_graph g3, rdf_graph g4
WHERE {
  ?x hasFather ?f .
  ?x hasMother ?m .
  ?y hasFather ?f .
  ?y hasMother ?m .
  g1.subject != g3.subject
}
```
SQL and graph query

Querying RDF data

- Find pairs of siblings (same parents)
- SELECT ?x, ?y FROM rdf_graph WHERE {
  ?x hasFather ?f .
  ?x hasMother ?m .
  ?y hasFather ?f .
  ?y hasMother ?m .
  FILTER(?x != ?y)}
Oracle Database 11g R2
Semantic Technologies:
Storage, Query, Enterprise RDF
Architectural Overview
Architectural Overview

- Enterprise (Relational) data
- RDF/OWL data and ontologies
- Rulebases: OWL, RDF/S, user-defined
- Inferred RDF/OWL data

Security: fine-grained
Versioning: Workspaces
Semantic Indexes
Architectural Overview

Core functionality:
- LOAD:
  - Bulk-Load
  - Incr. DML
- INFER:
  - OWL subsets
  - RDF/S
  - User-def.
- QUERY (SQL-based SPARQL):
  - Query RDF/OWL data and ontologies
  - Ontology-assisted Query of Enterprise Data

Oracle DB:
- RDF/OWL data and ontologies
- Rulebases: OWL, RDF/S, user-defined
- Inferred RDF/OWL data
- Semantic Indexes
- Enterprise (Relational) data

Security: fine-grained
Versioning: Workspaces
Architectural Overview

Programming Interface

- SPARQL (Jena)
- Java Programs
- JDBC
- Java API support

SQL Interface

- SQLplus
- PL/SQL
- SQLdev.

Core functionality

- LOAD
  - Bulk-Load
  - Incre. DML
- INFER
  - OWL subsets
  - RDF/S
  - User-def.
- QUERY (SQL-based SPARQL)
  - Query RDF/OWL data and ontologies
  - Ontology-assisted Query of Enterprise Data

Oracle DB

- RDF/OWL data and ontologies
- Rulebases: OWL, RDF/S, user-defined
- Inferred RDF/OWL data
- Semantic Indexes
- Enterprise (Relational) data

- Security: fine-grained
- Versioning: Workspaces
Architectural Overview

Programming Interface

- SPARQL (Jena)
- Java API support
- JDBC
- Java Programs
- SQL Interface
  - SQLplus
  - PL/SQL
  - SQLdev.

Core functionality

- LOAD
  - Bulk-Load
  - Incr. DML
- INFER
  - OWL subsets
  - RDF/S
  - User-def.
- QUERY (SQL-based SPARQL)
  - Query RDF/OWL data and ontologies
  - Ontology-assisted Query of Enterprise Data

Oracle DB

- RDF/OWL data and ontologies
- Rulebases: OWL, RDF/S, user-defined
- Inferred RDF/OWL data
- Security: fine-grained
- Versioning: Workspaces
- Semantic Indexes
- Enterprise (Relational) data

3rd-Party Callouts

Reasoners: Pellet
NLP Info. Extractor: Calais, GATE
Java API support
SPARQL (Jena)
Java Programs
SQL Interface
JDBC
SQLplus
PL/SQL
SQLdev.

Java Programs
SPARQL (Jena)
Java API support
JDBC
SQL Interface
JDBC
SQLplus
PL/SQL
SQLdev.
Architectural Overview

Programming Interface:
- SPARQL (Jena)
- Java Programs
- JDBC

SQL Interface:
- SQLplus
- PL/SQL
- SQLdev.

Java API support

Core functionality:
- LOAD: Bulk-Load, Incr. DML
- INFER: OWL subsets, RDF/S, User-def.

QUERY (SQL-based SPARQL):
- Query RDF/OWL data and ontologies
- Ontology-assisted Query of Enterprise Data

Oracle DB:
- RDF/OWL data and ontologies
- Rulebases: OWL, RDF/S, user-defined
- Inferred RDF/OWL data

Security: fine-grained
Versioning: Workspaces

Semantic Indexes

Enterprise (Relational) data

Tools:
- Visualizer (cytoscope)
- Reasoners: Pellet

3rd-Party Callouts:
- NLP Info. Extractor: Calais, GATE

Oracle
Architectural Overview

3rd-Party Tools
- Topbraid Composer
- SPARQL Endpoint (Joseki)

Programming Interface
- SPARQL (Jena)
- Java Programs
- Java API support
- JDBC

SQL Interface
- SQLplus
- PL/SQL
- SQLdev.

3rd-Party Tools
- Topbraid Composer
- SPARQL Endpoint (Joseki)

Core functionality
- LOAD
  - Bulk-Load
  - Incr. DML
- INFERENCE
  - OWL subsets
  - RDF/S
  - User-def.
- QUERY (SQL-based SPARQL)
  - Query RDF/OWL data and ontologies
  - Ontology-assisted Query of Enterprise Data

Oracle DB
- RDF/OWL data and ontologies
- Rulebases: OWL, RDF/S, user-defined
- Inferred RDF/OWL data

Security: fine-grained
Versioning: Workspaces
Semantic Indexes
Enterprise (Relational) data

NLP Info. Extractor: Calais, GATE
Visualizer (cytoscope)
Reasoners: Pellet

Programming Interface
Java API support
Java Programs
Core Entities in Oracle Database Semantic Store
Core Entities in Oracle Database Semantic Store

Core Entities in Oracle Database Semantic Store

- Sem. Network ➔ Dictionary and data tables. **New entities:** models, rule bases, and rules indexes (entailments). OWL and RDFS rule bases are preloaded.
Core Entities in Oracle Database Semantic Store

- Sem. Network ➔ Dictionary and data tables. **New entities:** models, rule bases, and rules indexes (entailments). OWL and RDFS rule bases are preloaded.
Core Entities in Oracle Database Semantic Store

- Sem. Network ➔ Dictionary and data tables. **New entities**: models, rule bases, and rules indexes (entailments). OWL and RDFS rule bases are preloaded.
Core Entities in Oracle Database Semantic Store

- Sem. Network ➔ Dictionary and data tables. **New entities**: models, rule bases, and rules indexes (entailments). OWL and RDFS rule bases are preloaded.

- **SDO_RDF_TRIPLE_S** ➔ A new object type for RDF.

<table>
<thead>
<tr>
<th>Rulebases &amp; Vocabularies</th>
</tr>
</thead>
<tbody>
<tr>
<td>OWL subset</td>
</tr>
<tr>
<td>RDF / RDFS</td>
</tr>
</tbody>
</table>

Semantic Network (MDSYS)

Triples

Values
Core Entities in Oracle Database Semantic Store

- **Sem. Network ➔ Dictionary and data tables.** New entities: models, rule bases, and rules indexes (entailments). OWL and RDFS rule bases are preloaded.

- **SDO_RDF_TRIPLE_S ➔ A new object type for RDF.**

- **Application Table ➔** Contains col of object type sdo_rdf_triple_s to allow loading and accessing RDF triples, and storing ancillary values.
Core Entities in Oracle Database Semantic Store

- **Sem. Network** ➔ Dictionary and data tables. **New entities:** models, rule bases, and rules indexes (entailments). OWL and RDFS rule bases are preloaded.

- **SDO_RDF_TRIPLE_S** ➔ A new object type for RDF.

- **Application Table** ➔ Contains col of object type sdo_rdf_triple_s to allow loading and accessing RDF triples, and storing ancillary values.

- **Model** ➔ A model holds an RDF graph (set of S-P-O triples) and is associated with an sdo_rdf_triple_s column in an application table.
Core Entities in Oracle Database Semantic Store

- **Sem. Network** ➔ Dictionary and data tables. **New entities**: models, rule bases, and rules indexes (entailments). OWL and RDFS rule bases are preloaded.

- **SDO_RDF_TRIPLE_S** ➔ A new object type for RDF.

- **Application Table** ➔ Contains col of object type sdo_rdf_triple_s to allow loading and accessing RDF triples, and storing ancillary values.

- **Model** ➔ A model holds an RDF graph (set of S-P-O triples) and is associated with an sdo_rdf_triple_s column in an application table.

- **Rulebase** ➔ A rulebase is a set of rules used for inferencing.
Core Entities in Oracle Database Semantic Store

- **Sem. Network** ➔ Dictionary and data tables. **New entities**: models, rule bases, and rules indexes (entailments). OWL and RDFS rule bases are preloaded.

- **SDO_RDF_TRIPLE_S** ➔ A new object type for RDF.

- **Application Table** ➔ Contains col of object type sdo_rdf_triple_s to allow loading and accessing RDF triples, and storing ancillary values.

- **Model** ➔ A model holds an RDF graph (set of S-P-O triples) and is associated with an sdo_rdf_triple_s column in an application table.

- **Rulebase** ➔ A rulebase is a set of rules used for inferencing.

- **Entailments** ➔ An entailment stores triples derived via inferencing.
Network Indexes

- Some B-tree indexes (including one for enforcing uniqueness constraint) are created automatically at sem. network creation time.
- The set of usable indexes can be manipulated using
  - `sem_apis.add_sem_index`
  - `sem_apis.drop_sem_index`
  - `sem_apis.alter_index_on_model`
  - `sem_apis.alter_index_on_entailment`
- *Tip*: when creating a network index, use the `model_id` column as part (usually, the last col) of the index key
Virtual Model

- Similar to simple database Views
- One may define a Virtual Model consisting of multiple sem. Models, and optionally the corresponding entailment.
- The relevant subprograms are
  - `sem_apis.create_virtual_model`
  - `sem_apis.drop_virtual_model`
- Benefits
  - Allows ease and flexibility in specifying target models plus (optionally) the corresponding entailment
  - Facilitates access control
  - Query performance (esp. for queries against multi-model, or single/multi-model+entailment RDF data source)
Usage flow (via SQL interface)

Create “Application Table” and model

- create table app_table (triple sdo_rdf_triple_s);
- exec sem_apis.create_sem_model (‘family’, ’app_table’, ’triple’);

Load data

- insert into app_table (triple) values(sdo_rdf_triple_s(‘family’,…))
  - Use sem_apis.bulk_load_from_staging_table() for large loads
- Collect statistics using: exec sem_apis.analyze_model(‘family’);

Inference

- exec sem_apis.create_entailment (‘family_owl_ent’, sem_models(‘family’),
  sem_rulebases(‘owlprime’));
- Collect statistics using: exec sem_apis.analyze_rules_index(‘family_owl_ent’);

Query

- select x, y from table(sem_match(‘{?x rdf:type ?y}’, sem_models (‘family’),
  sem_rulebases(‘owlprime’), null, null));
SPARQL in SQL

```sql
{?x hasFather ?f .
 ?x hasMother ?m .
 ?y hasFather ?f .
 ?y hasMother ?m .
 FILTER( ?x != ?y ) }
```
SPARQL in SQL

Querying RDF data

• Find pairs of siblings (same parents)
  
  SELECT x, y
  FROM
  WHERE
{?x hasFather ?f .
  ?x hasMother ?m .
  ?y hasFather ?f .
  ?y hasMother ?m .
  FILTER( ?x != ?y )}
SPARQL in SQL

Querying RDF data

- Find pairs of siblings (same parents)
- SELECT x, y
  FROM
  WHERE
  {?
x hasFather ?f.
  ?x hasMother ?m.
  ?y hasFather ?f.
  ?y hasMother ?m.
  FILTER( ?x != ?y )}
SPARQL in SQL

Querying RDF data

• Find pairs of siblings (same parents)
• SELECT x, y
FROM WHERE
{
  ?x hasFather ?f .
  ?x hasMother ?m .
  ?y hasFather ?f .
  ?y hasMother ?m .
  FILTER( ?x != ?y )
}

Evaluates to

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Suzie</td>
<td>Matt</td>
</tr>
<tr>
<td>Matt</td>
<td>Suzie</td>
</tr>
<tr>
<td>Cathy</td>
<td>Jack</td>
</tr>
<tr>
<td>Jack</td>
<td>Cathy</td>
</tr>
<tr>
<td>Tom</td>
<td>Cindy</td>
</tr>
<tr>
<td>Cindy</td>
<td>Tom</td>
</tr>
</tbody>
</table>
SPARQL in SQL

**Querying RDF data**

- Find pairs of siblings (same parents)
- SELECT x, y
- FROM
- WHERE

```sparql
{?
x hasFather ?f .
?x hasMother ?m .
?y hasFather ?f .
?y hasMother ?m .
FILTER( ?x != ?y ) }
```

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Suzie</td>
<td>Matt</td>
</tr>
<tr>
<td>Matt</td>
<td>Suzie</td>
</tr>
<tr>
<td>Cathy</td>
<td>Jack</td>
</tr>
<tr>
<td>Jack</td>
<td>Cathy</td>
</tr>
<tr>
<td>Tom</td>
<td>Cindy</td>
</tr>
<tr>
<td>Cindy</td>
<td>Tom</td>
</tr>
</tbody>
</table>

Allows use of rich SQL constructs such as: aggregates, subqueries, analytical functions, expressions, etc. and joining with object-relational data (e.g., spatial).
Query performance: Rewrite to SQL

- Rewriting the **SPARQL in SQL** to SQL allows
  - avoiding communication of data between SQL engine and non-SQL engine, and
  - making the whole query visible to the RDBMS optimizer as a single SQL query (thereby maximizing the possibility of getting the best execution plan chosen by the optimizer)
Query performance: Rewrite to SQL

• Rewriting the SPARQL in SQL to SQL allows
  – avoiding communication of data between SQL engine and non-SQL engine, and
  – making the whole query visible to the RDBMS optimizer as a single SQL query (thereby maximizing the possibility of getting the best execution plan chosen by the optimizer)

Querying RDF data

• Find pairs of siblings (same parents)
• SELECT x, y
FROM
WHERE
Query performance: Rewrite to SQL

• Rewriting the SPARQL in SQL to SQL allows
  – avoiding communication of data between SQL engine and non-SQL engine, and
  – making the whole query visible to the RDBMS optimizer as a single SQL query (thereby maximizing the possibility of getting the best execution plan chosen by the optimizer)

---

Querying RDF data

• Find pairs of siblings (same parents)

• SELECT x, y
  FROM WHERE

```sparql
{?x hasFather ?f . ?x hasMother ?m . ?y hasFather ?f . ?y hasMother ?m . FILTER( ?x != ?y ) }
```
Querying RDF data

- Find pairs of siblings (same parents)

```sql
SELECT x, y
FROM
WHERE
{ ?x hasFather ?f .
  ?x hasMother ?m .
  ?y hasFather ?f .
  ?y hasMother ?m .
  FILTER( ?x != ?y ) }
```

Rewriting the SPARQL in SQL to SQL allows
- avoiding communication of data between SQL engine and non-SQL engine, and
- making the whole query visible to the RDBMS optimizer as a single SQL query (thereby maximizing the possibility of getting the best execution plan chosen by the optimizer)
Query performance: Rewrite to SQL

- Rewriting the **SPARQL in SQL** to SQL allows
  - avoiding communication of data between SQL engine and non-SQL engine, and
  - making the whole query visible to the RDBMS optimizer as a single SQL query (thereby maximizing the possibility of getting the best execution plan chosen by the optimizer)

### Querying RDF data
- Find pairs of siblings (same parents)
- `SELECT x, y
  FROM
  WHERE
  {?x hasFather ?f .
   ?x hasMother ?m .
   ?y hasFather ?f .
   ?y hasMother ?m .
   FILTER( ?x != ?y ) }

  SELECT ...
  FROM ...
  WHERE ...`
Querying RDF data in SQL using SEM_MATCH

Querying RDF data
- Find pairs of siblings (same parents)
- SELECT ?x ?y
  FROM <family> WHERE {
    ?x hasFather ?f . ?x hasMother ?m .
    ?y hasFather ?f . ?y hasMother ?m .
    FILTER( ?x != ?y )
  }

Querying RDF data in Oracle 11g
- Find pairs of siblings (same parents)
  SELECT x, y
  FROM TABLE(SEM_MATCH(‘{
    ?x :hasFather ?f . ?x :hasMother ?m .
    ?y :hasFather ?f . ?y :hasMother ?m .
    FILTER(?x != ?y)
  }’,
  sem_models(‘family’),null,
  sem_aliases(sem_alias(“,’http://f.com/‘)),
  null))
  ;
Querying RDF data in SQL using SEM_MATCH

- **SEM_MATCH** rewriterable table function

```
• Querying RDF data
• Find pairs of siblings (same parents)
• SELECT ?x ?y
  FROM <family> WHERE {
    ?x hasFather ?f . ?x hasMother ?m .
    ?y hasFather ?f . ?y hasMother ?m .
    FILTER( ?x != ?y )
  }
```

```
Querying RDF data in Oracle 11g
• Find pairs of siblings (same parents)
SELECT x, y
FROM
TABLE(SEM_MATCH("{
  ?x :hasFather ?f . ?x :hasMother ?m .
  ?y :hasFather ?f . ?y :hasMother ?m .
  FILTER(?x != ?y)
}"
),
  sem_models(‘family’),null,
  sem_aliases(sem_alias(“”, ‘http://f.com/’),null))
);```
Querying RDF data in SQL using SEM_MATCH

- **SEM_MATCH** rewriterable table function

- SELECT ...
  FROM TABLE(SEM_MATCH(
    pattern, models, rulebases, aliases ...)
  )
  WHERE ...

Querying RDF data

- Find pairs of siblings (same parents)
- SELECT ?x ?y
  FROM <family> WHERE {
    ?x hasFather ?f . ?x hasMother ?m .
    ?y hasFather ?f . ?y hasMother ?m .
    FILTER( ?x != ?y )
  }

Querying RDF data in Oracle 11g

- Find pairs of siblings (same parents)
- SELECT x, y
  FROM TABLE(SEM_MATCH('{
    ?x :hasFather ?f . ?x :hasMother ?m .
    ?y :hasFather ?f . ?y :hasMother ?m .
    FILTER(?x != ?y)
  }',
  sem_models('family'),null,
  sem_aliases(sem_alias('","http://f.com/"'),
  null))
)
Querying RDF data in SQL using SEM_MATCH

- **SEM_MATCH** rewritable table function

- `SELECT ... FROM TABLE(SEM_MATCH(
  pattern, models, rulebases, aliases...)) t ...
WHERE ...`

- SPARQL graph pattern *embedded* in SQL query

---

Querying RDF data

- Find pairs of siblings (same parents)
- `SELECT ?x ?y FROM <family> WHERE {
  ?x hasFather ?f . ?x hasMother ?m .
  ?y hasFather ?f . ?y hasMother ?m .
  FILTER(?x != ?y) }

---

Querying RDF data in Oracle 11g

- Find pairs of siblings (same parents)
- `SELECT x, y FROM TABLE(SEM_MATCH(`
  '{
    ?x :hasFather ?f . ?x :hasMother ?m .
    ?y :hasFather ?f . ?y :hasMother ?m .
    FILTER(?x != ?y)
  }',
  sem_models('family'),null,
  sem_aliases(sem_alias('"','http://f.com/"),
  null))
);`
Querying RDF data in SQL using SEM_MATCH

- **SEM_MATCH** *rewritable* table function

- SELECT ...
  FROM TABLE(SEM_MATCH(
    pattern,models,rulebases,
    aliases...) t ...
  WHERE ...

- SPARQL graph pattern *embedded* in SQL query

- Matches specified graph pattern against stored data
Querying RDF data in SQL using SEM_MATCH

- **SEM_MATCH** rewritable table function

- `SELECT ... FROM TABLE(SEM_MATCH( pattern, models, rulebases, aliases... ) t ... WHERE ...`

- SPARQL graph pattern *embedded* in SQL query

- Matches specified graph pattern against stored data

- Returns a *table* of results: Each result row is values for selected columns corresponding to mappings for the graph pattern variables

---

Querying RDF data

- Find pairs of siblings (same parents)
- `SELECT ?x ?y FROM <family> WHERE { ?x hasFather ?f . ?x hasMother ?m . ?y hasFather ?f . ?y hasMother ?m . FILTER(?x != ?y) }

---

Querying RDF data in Oracle 11g

- Find pairs of siblings (same parents)
- `SELECT x, y FROM TABLE(SEM_MATCH( '{
  ?x :hasFather ?f . ?x :hasMother ?m .
  ?y :hasFather ?f . ?y :hasMother ?m .
  FILTER(?x != ?y)
}',
  sem_models('family'), null,
  sem_aliases(sem_alias('“', 'http://f.com/’)),
  null))

;`
HINT framework: Aliases
HINT framework: Aliases

```
SELECT e, loc FROM
   TABLE(SEM_MATCH( '{?e rdf:type :SoftwareEngineer .
                     ?e :worksFor ?company .
                     ?company :headQuarterLoc ?loc}',
   SEM_Models('emp', 'gmap'), SEM_Rulebases('OWLPRIME'),
   SEM_ALIASES(SEM_ALIAS('', 'http://example.org/')),NULL));
```
SELECT e, loc FROM TABLE(SEM_MATCH( "{?e rdf:type :SoftwareEngineer .
    ?e :worksFor ?company .
    ?company :headQuarterLoc ?loc}",
    SEM_Models('emp', 'gmap'), SEM_Rulebases('OWLPRIME'),
    SEM_ALIASES(SEM_ALIAS('', 'http://example.org/')),NULL));
HINT framework: Aliases

```
SELECT e, loc FROM TABLE(SEM_MATCH( '{?e rdf:type :SoftwareEngineer .
    ?e :worksFor ?company .
    ?company :headQuarterLoc ?loc}',
    SEM_Models('emp', 'gmap'), SEM_Rulebases('OWLPRIME'),
    SEM_ALIASES(SEM_ALIAS('', 'http://example.org/')),NULL));
```
HINT framework: Aliases

```sql
SELECT e, loc FROM
  TABLE(SEM_MATCH( '{?e rdf:type :SoftwareEngineer .
    ?e :worksFor ?company .
    ?company :headQuarterLoc ?loc}')
  SEM_Models('emp', 'gmap'), SEM_Rulebases('OWLPRIME'),
  SEM_ALIASES(SEM_ALIAS('', 'http://example.org/')),NULL));
```
HINT framework: Aliases

```
SELECT e, loc FROM
TABLE(SEM_MATCH( '{?e rdf:type :SoftwareEngineer .
    ?e :worksFor ?company .
    ?company :headQuarterLoc ?loc}'), SEM_Models('emp', 'gmap'), SEM_Rulebases('OWLPRIME'), SEM_ALIASES(SEM_ALIAS('', 'http://example.org/')), NULL));
```
HINT framework: Aliases

SELECT e, loc FROM TABLE(SEM_MATCH( '{?e rdf:type :SoftwareEngineer .
   ?e :worksFor ?company .
   ?company :headQuarterLoc ?loc}';
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 ⇒ aliases for RDF Model
  – For the 3 distinct variables:
    ?e, ?company, ?loc ⇒ aliases for RDF_VALUE$
HINT framework: Aliases

SELECT e, loc FROM TABLE(SEM_MATCH( '{?e rdf:type :SoftwareEngineer .
    ?e :worksFor ?company .
    ?company :headQuarterLoc ?loc}’
    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 3 distinct variables:
    ?e, ?company, ?loc \(\rightarrow\) aliases for RDF_VALUE$

• 6-way Join query:
  – t0 x t1 x t2 x ?e x ?company x ?loc
HINT framework: Aliases

SELECT e, loc FROM TABLE(SEM_MATCH('{$e rdf:type :SoftwareEngineer .
    ?e :worksFor ?company .
    ?company :headQuarterLoc ?loc}'), 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 3 distinct variables: ?e, ?company, ?loc \rightarrow aliases for RDF_VALUE$

- 6-way Join query:
  - t0 x t1 x t2 x ?e x ?company x ?loc

- Use of *projection optimization* reduces it to a 5-way join query since columns for only the ?e and ?loc variables are referred in containing SQL query: t0 x t1 x t2 x ?e x ?loc
• HINTS in SEM_MATCH query
  – Similar to SQL hints
    • Join order: LEADING(...), ORDERED
    • Join type: USE_HASH(...), USE_NL(...), ...
    • Access path: INDEX(...), ...
  – Aliases are a bit different
    • Variable names: ?e, ?company, ?loc, ...
    • Pattern ordinal based: t0, t1, t2, ...
HINT specification

SELECT e, loc FROM
TABLE(SEM_MATCH( '{?e rdf:type :SoftwareEngineer .
  ?e :worksFor ?company .
  ?company :headQuarterLoc ?loc}',
  SEM_Models('emp', 'gmap'), SEM_Rulebases('OWLPRIME'),
  SEM_ALIASES(SEM_ALIAS('', 'http://example.org/')),NULL,NULL,
  ' HINT0 = { LEADING(t0 t1 t2) USE_NL(t1) } '));

- **HINTS in SEM_MATCH query**
  - Similar to SQL hints
    - Join order: **LEADING(…), ORDERED**
    - Join type: **USE_HASH(…), USE_NL(…), …**
    - Access path: **INDEX(…), …**
  - Aliases are a bit different
    - Variable names: ?e, ?company, ?loc, ...
    - Pattern ordinal based: t0, t1, t2, …
Hint specification

SELECT e, loc FROM TABLE(SEM_MATCH( ‘{?e rdf:type :SoftwareEngineer .
   ?e :worksFor ?company .
   ?company :headQuarterLoc ?loc}’
SEM_Models(‘emp’, ‘gmap’), SEM_Rulebases(‘OWLPRIME’),
SEM_ALIASES(SEM_ALIAS('', 'http://example.org/')),NULL,NULL,
‘ HINT0 = { LEADING(t0 t1 t2) USE_NL(t1)} ‘));

- **HINTS in SEM_MATCH query**
  - Similar to SQL hints
    - Join order: **LEADING(...), ORDERED**
    - Join type: **USE_HASH(...), USE_NL(...), ...**
    - Access path: **INDEX(...), ...**
  - Aliases are a bit different
    - Variable names: ?e, ?company, ?loc, ...
    - Pattern ordinal based: t0, t1, t2, ...
Hint specification

**HINTS in SEM_MATCH query**
- Similar to SQL hints
  - Join order: `LEADING(...)`, `ORDERED`
  - Join type: `USE_HASH(...)`, `USE_NL(...)`, ...
  - Access path: `INDEX(...)`, ...
- Aliases are a bit different
  - Variable names: `?e`, `?company`, `?loc`, ...
  - Pattern ordinal based: `t0`, `t1`, `t2`, ...

```sql
SELECT e, loc FROM TABLE(SEM_MATCH(
    '{?e rdf:type :SoftwareEngineer .
     ?e :worksFor ?company .
     ?company :headQuarterLoc ?loc}"
    SEM_Models('emp', 'gmap'),
    SEM_Rulebases('OWLPRIME'),
    SEM_ALIASES(SEM_ALIAS('', 'http://example.org/')),
    ' HINT0 = { LEADING(t0 t1 t2) USE_NL(t1) } ');)
```
Ontology-assisted Query using SQL 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>
```

Syntactic match returns no rows:
```
SELECT p_id, diagnosis FROM Patients WHERE diagnosis = 'Upper_Extremity_Fracture';
```
Ontology-assisted Query using SQL Operators

```
SELECT p_id, diagnosis
FROM Patients
WHERE SEM_RELATED (diagnosis, 'rdfs:subClassOf', 'Upper_Extremity_Fracture', sem_models('Medical_ontology'), sem_rulebases('RDFS') …) = 1;
```
Ontology-assisted Query using SQL Operators

SELECT p_id, diagnosis
FROM Patients
WHERE SEM_RELATED (diagnosis, ‘rdfs:subClassOf’, ‘Upper_Extremity_Fracture’, sem_models(‘Medical_ontology’), sem_rulebases(‘RDFS’) ... 123) = 1
AND SEM_DISTANCE(123) <= 2;
Indiana authorities filed felony charges and a court issued an arrest warrant for a financial manager who apparently tried to fake his death by crashing his airplane in a Florida swamp. Marcus Schrenker, 38 …

<table>
<thead>
<tr>
<th>docId</th>
<th>Article</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Indiana authorities filed felony charges and a court issued an arrest warrant for a financial manager who apparently tried to fake his death by crashing his airplane in a Florida swamp. Marcus Schrenker, 38 …</td>
<td>CNN</td>
</tr>
<tr>
<td>2</td>
<td>Major dealers and investors …</td>
<td>NW</td>
</tr>
<tr>
<td>..</td>
<td>..</td>
<td>..</td>
</tr>
</tbody>
</table>
Indiana authorities filed felony charges and a court issued an arrest warrant for a financial manager who apparently tried to fake his death by crashing his airplane in a Florida swamp. Marcus Schrenker, 38 …

```
CREATE INDEX ArticleIndex
ON NewsFeed (Article)
INDEXTYPE IS SemContext
PARAMETERS ('gate_nlp')
```

**Newsfeed table**

<table>
<thead>
<tr>
<th>docId</th>
<th>Article</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Indiana authorities filed felony charges and a court issued an arrest warrant for a financial manager who apparently tried to fake his death by crashing his airplane in a Florida swamp. Marcus Schrenker, 38 …</td>
<td>CNN</td>
</tr>
<tr>
<td>2</td>
<td>Major dealers and investors …</td>
<td>NW</td>
</tr>
<tr>
<td></td>
<td>..</td>
<td>..</td>
</tr>
</tbody>
</table>
Indiana authorities filed felony charges and a court issued an arrest warrant for a financial manager who apparently tried to fake his death by crashing his airplane in a Florida swamp. Marcus Schrenker, 38 …

<table>
<thead>
<tr>
<th>docId</th>
<th>Article</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Indiana authorities filed felony charges and a court issued an arrest warrant for a financial manager who apparently tried to fake his death by crashing his airplane in a Florida swamp. Marcus Schrenker, 38 …</td>
<td>CNN</td>
</tr>
<tr>
<td>2</td>
<td>Major dealers and investors …</td>
<td>NW</td>
</tr>
<tr>
<td></td>
<td>..</td>
<td>..</td>
</tr>
</tbody>
</table>
Indiana authorities filed felony charges and a court issued an arrest warrant for a financial manager who apparently tried to fake his death by crashing his airplane in a Florida swamp. Marcus Schrenker, 38 …

Major dealers and investors …

CREATE INDEX ArticleIndex ON NewsFeed (Article) INDEXTYPE IS SemContext PARAMETERS ('gate_nlp')

<table>
<thead>
<tr>
<th>Subject</th>
<th>Property</th>
<th>Object</th>
<th>rid</th>
</tr>
</thead>
<tbody>
<tr>
<td>p:Marcus</td>
<td>rdf:type</td>
<td>rc::Person</td>
<td>r1</td>
</tr>
<tr>
<td>p:Marcus</td>
<td>:hasName</td>
<td>&quot;Marcus&quot;^^...</td>
<td>r1</td>
</tr>
<tr>
<td>p:Marcus</td>
<td>:hasAge</td>
<td>&quot;38&quot;^^xsd:...</td>
<td>r1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>docId</th>
<th>Article</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Indiana authorities filed felony charges and a court issued an arrest warrant for a financial manager who apparently tried to fake his death by crashing his airplane in a Florida swamp. Marcus Schrenker, 38 …</td>
<td>CNN</td>
</tr>
<tr>
<td>2</td>
<td>Major dealers and investors …</td>
<td>NW</td>
</tr>
<tr>
<td>..</td>
<td>..</td>
<td>..</td>
</tr>
</tbody>
</table>
Indiana authorities filed felony charges and a court issued an arrest warrant for a financial manager who apparently tried to fake his death by crashing his airplane in a Florida swamp. Marcus Schrenker, 38 ...

```sql
SELECT docId FROM Newsfeed WHERE SEM_CONTAINS (Article, '{?x rdf:type rc:Person . ?x :hasAge ?age . FILTER(?age >= 35)}')=1
```
Indiana authorities filed felony charges and a court issued an arrest warrant for a financial manager who apparently tried to fake his death by crashing his airplane in a Florida swamp. Marcus Schrenker, 38 …

SELECT docId FROM Newsfeed
WHERE SEM_CONTAINS (Article, '{?x rdf:type rc:Person . ?x :hasAge ?age . FILTER(?age >= 35)}')=1
AND Source = ‘CNN’
Semantic Indexing of Documents

CREATE INDEX ArticleIndex
ON NewsFeed (Article)
INDEXTYPE IS SemContext
PARAMETERS (‘gate_nlp’)

CREATE INDEX ArticleIndex
ON NewsFeed (Article)
INDEXTYPE IS SemContext
PARAMETERS (‘gate_nlp’)

SELECT docId FROM Newsfeed
WHERE SEM_CONTAINS (Article,
    ‘{?x rdf:type rc:Person .
    ?x :hasAge ?age .
    FILTER(?age >= 35)}’) = 1
AND Source = ‘CNN’

Newsfeed table

<table>
<thead>
<tr>
<th>docId</th>
<th>Article</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Indiana authorities filed felony charges and a court issued an arrest warrant for a financial manager who apparently tried to fake his death by crashing his airplane in a Florida swamp. Marcus Schrenken, 38 ...</td>
<td>CNN</td>
</tr>
<tr>
<td>2</td>
<td>Major dealers and investors ...</td>
<td>NW</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

Analytical Queries

On Graph Data

11.2
[VPD] Access control policies on semantic data
Access control policies on semantic data

**Policy** ➔ user can only access active projects

**Match pattern** ➔ `{ ?x rdf:type :Project }`

**Apply pattern** ➔ `{ ?x :hasStatus :Active }`
[VPD] Access control policies on semantic data

**Policy** ➔ user can only access active projects
**Match pattern** ➔ { ?x rdf:type :Project } 
**Apply pattern** ➔ { ?x :hasStatus :Active } 

Query: Get the list of projects and their values

```
SELECT  ?proj  ?val
FROM    ProjectsGraph
WHERE   { ?proj   :hasValue  ?val }
```
Access control policies on semantic data

Policy ➔ user can only access active projects

Match pattern ➔ { ?x rdf:type :Project }

Apply pattern ➔ { ?x :hasStatus :Active }

Query: Get the list of projects and their values

SELECT ?proj ?val
FROM ProjectsGraph
WHERE { ?proj :hasValue ?val .
        ?proj :hasStatus :Active }
[VPD] Access control policies on semantic data

Policy ➔ user can access value of projects s/he leads
Match pattern ➔ { ?x :hasValue ?v }
Apply pattern ➔ { ?x :hasLead "sys_context(...)" }

Query : Get the list of projects and their values

```
SELECT ?proj ?val
FROM ProjectsGraph
WHERE { ?proj :hasValue ?val }
```
[VPD] Access control policies on semantic data

**Policy** ➔ user can access value of projects s/he leads

**Match pattern** ➔ \{ ?x :hasValue ?v \}

**Apply pattern** ➔ \{ ?x :hasLead "sys_context(...)" \}

---

**Query**: Get the list of projects and their values

```
SELECT ?proj ?val
FROM ProjectsGraph
WHERE { ?proj :hasValue ?val .
  ?proj :hasLead "sys_context(...)" }
```
Access control policies on semantic data

**Query**: Get the list of projects and their values

```
SELECT ?proj ?val
FROM ProjectsGraph
WHERE { ?proj :hasValue ?val }
```
[OLS] Data classification labels for semantic data

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
<th>Row Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectHLS</td>
<td>Organization</td>
<td>N.America</td>
<td>SE:HLS:US</td>
</tr>
<tr>
<td>projectHLS</td>
<td>ContractValue</td>
<td>1000000</td>
<td>SE:HLS,FIN:US</td>
</tr>
</tbody>
</table>
[OLS] Data classification labels for semantic data

SELECT operation: Labels for triples are used to restrict access to the triples.

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
<th>Row Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectHLS</td>
<td>Organization</td>
<td>N.America</td>
<td>SE:HLS:US</td>
</tr>
<tr>
<td>projectHLS</td>
<td>ContractValue</td>
<td>1000000</td>
<td>SE:HLS,FIN:US</td>
</tr>
</tbody>
</table>
[OLS] Data classification labels for semantic data

SELECT operation: Labels for triples are used to restrict access to the triples.

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
<th>Row Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectHLS</td>
<td>Organization</td>
<td>N.America</td>
<td>SE:HLS:US</td>
</tr>
<tr>
<td>projectHLS</td>
<td>ContractValue</td>
<td>1000000</td>
<td>SE:HLS,FIN:US</td>
</tr>
</tbody>
</table>

INSERT operation: Labels for resources to control their use as component of a triple during insertion. (Also, determines min. access label for inserted triple.)
SELECT operation: Labels for triples are used to restrict access to the triples.

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
<th>Row Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectHLS</td>
<td>Organization</td>
<td>N.America</td>
<td>SE:HLS:US</td>
</tr>
<tr>
<td>projectHLS</td>
<td>ContractValue</td>
<td>1000000</td>
<td>SE:HLS,FIN:US</td>
</tr>
</tbody>
</table>

INSERT operation: Labels for resources to control their use as component of a triple during insertion. (Also, determines min. access label for inserted triple.)
[OLS] Data classification labels for semantic data

**SELECT operation:** Labels for triples are used to restrict access to the triples.

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
<th>Row Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectHLS</td>
<td>Organization</td>
<td>N.America</td>
<td>SE:HLS:US</td>
</tr>
<tr>
<td>projectHLS</td>
<td>ContractValue</td>
<td>1000000</td>
<td>SE:HLS,FIN:US</td>
</tr>
</tbody>
</table>

**INSERT operation:** Labels for resources to control their use as component of a triple during insertion. (Also, determines min. access label for inserted triple.)

<table>
<thead>
<tr>
<th>Position &amp; Resource</th>
<th>Row Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subject: projectHLS</td>
<td>SE:HLS:US</td>
</tr>
</tbody>
</table>
[OLS] Data classification labels for semantic data

**SELECT operation:** Labels for triples are used to restrict access to the triples.

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
<th>Row Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectHLS</td>
<td>Organization</td>
<td>N.America</td>
<td>SE:HLS:US</td>
</tr>
<tr>
<td>projectHLS</td>
<td>ContractValue</td>
<td>1000000</td>
<td>SE:HLS:FIN:US</td>
</tr>
</tbody>
</table>

**INSERT operation:** Labels for resources to control their use as component of a triple during insertion. (Also, determines min. access label for inserted triple.)

<table>
<thead>
<tr>
<th>Position &amp; Resource</th>
<th>Row Label</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Subject:</strong></td>
<td>projectHLS</td>
</tr>
<tr>
<td><strong>Predicate:</strong></td>
<td>hasContractValue</td>
</tr>
</tbody>
</table>
Data classification labels for semantic data

**SELECT operation**: Labels for triples are used to restrict access to the triples.

**INSERT operation**: Labels for resources to control their use as component of a triple during insertion. (Also, determines min. access label for inserted triple.)

### Triples table

<table>
<thead>
<tr>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
<th>Row Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>projectHLS</td>
<td>Organization</td>
<td>N.America</td>
<td>SE:HLS:US</td>
</tr>
<tr>
<td>projectHLS</td>
<td>ContractValue</td>
<td>1000000</td>
<td>SE:HLS:FIN:US</td>
</tr>
</tbody>
</table>

### Position & Resource

<table>
<thead>
<tr>
<th>Subject:</th>
<th>projectHLS</th>
<th>Row Label: SE:HLS:US</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predicate:</td>
<td>hasContractValue</td>
<td>TS:FIN:</td>
</tr>
</tbody>
</table>
Change Mgmt./Versioning for Sem. Data

- Manage public and private versions of semantic data in database workspaces (*Workspace Manager*).

- An RDF Model is *version-enabled* by version-enabling the corresponding application table.
  - `exec DBMS_WM.enableVersioning (table_name => 'contracts_rdf_data');`
- RDF data modified within a workspace is private to the workspace until it is merged.
- SEM_MATCH queries on version-enabled models are version aware and only return relevant data.
- New versions created only for changed data.
- Versioning is provisioned for inference.
Inference,
Jena Adaptor for Oracle Database,
Utilities, and Performance
Inference
Infer Semantic Data

• Native inferencing in the database for
  • RDF, RDFS, and a rich subset of OWL semantics (OWLSIF, OWLPRIME, RDFS++)
  • User-defined rules
• Forward chaining.
  • New relationships/triples are inferred and stored ahead of query time
  • Removes on-the-fly reasoning and results in fast query times
• Proof generation
  • Show one deduction path
Infer Semantic Data: APIs

**Recommended API for inference**

- **SEM_APICREATE_ENTAILMENT**
  - Index_name
  - sem_models(‘GraphTBox’, ‘GraphABox’, …),
  - sem_rulebases(‘OWLPrime’),
  - passes,
  - Inf_components,
  - Options

  Use “PROOF=T” to generate inference proof

- **SEM_API Validate_ENTAI LMENT**
  - sem_models(‘GraphTBox’, ‘GraphABox’, …),
  - sem_rulebases(‘OWLPrime’),
  - Criteria,
  - Max_conflict,s,
  - 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: GraphOracleSem.performInference()**

Typical Usage:
- First load RDF/OWL data
- Call create_entailment to generate inferred graph
- Call validate_entailment to find inconsistencies
New Release 11.2 Inference Features

- Richer semantics support including
  - `owl:intersectionOf`, `unionOf`, `oneOf`,
  - W3C SKOS
  - SNOMED
  - OWL2’s `owl:propertyChainAxiom`, `owl:NegativePropertyAssertion`, `owl:hasKey`, `owl:propertyDisjointWith`,
- Most OWL 2 RL/RDF rules

- Performance enhancement
  - Large scale `owl:sameAs` handling
  - Parallel inference
  - Incremental inference
New Release 11.2 Inference Components

- **UNION**: (OWL 1) owl:unionOf
- **INTERSECT & INTERSECTSCOH**: (OWL 1) owl:intersectionOf
- **SNOMED**: (OWL 2) Systematized Nomenclature of Medicine
- **PROPDISJH**: (OWL 2) interaction between owl:propertyDisjointWith and rdfs:subPropertyOf.
- **CHAIN**: (OWL 2) Supports chains of length 2
- **SKOSAXIOMS**: most of the axioms defined in SKOS reference
- **MBRLST**: for any resource, every item in the list given as the value of the skos:memberList property is also a value of the skos:member property.
- **SVFH**: capturing interaction between owl:someValuesFrom and rdfs:subClassOf
- **THINGH & THINGSAM**: any defined OWL class is a subclass of owl:Thing & instances of owl:Thing are equal to themselves
Enabling New Release 11.2 Inference Capabilities

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

- Enabling Incremental inference option
  ```java
  EXECUTE sem_apis.create_entailment ('M_IDX',sem_models('M'),
  sem_rulebases('OWLPRIME'),null,null, 'INC=T');
  ```

- Enabling owl:sameAs option to limit duplicates
  ```java
  EXECUTE Sem_apis.create_entailment('M_IDX',sem_models('M'),
  sem_rulebases('OWLPRIME'),null,null,'OPT_SAMEAS=T');
  ```

- Enabling compact data structures
  ```java
  EXECUTE Sem_apis.create_entailment('M_IDX',sem_models('M'),
  sem_rulebases('OWLPRIME'),null,null,'RAW8=T');
  ```

- Enabling SKOS inference
  ```java
  EXECUTE Sem_apis.create_entailment('M_IDX',sem_models('M'),
  sem_rulebases('SKOSCORE'),null,null...);
  ```
Jena Adaptor for Oracle Database 11g Release 2
Jena Adaptor for Oracle Database 11g Release 1

- Implements Jena’s Graph/Model/BulkUpdateHandler/… APIs
- “Proxy” like design
  - Data not cached in memory for scalability
  - SPARQL query converted into SQL and executed inside DB
    - A SPARQL with just conjunctive patterns is converted into a single SEM_MATCH query
- Allows various data loading
  - Bulk/Batch/Incremental load RDF or OWL (in N3, RDF/XML, N-TRIPLE etc.) with strict syntax verification and long literal support
- Integrates Oracle Database release 11g RDF/OWL with tools including
  - TopBraid Composer
  - External complete DL reasoners (e.g. Pellet)

Programming Semantic Applications in Java

• Create an Oracle object
  - oracle = new Oracle(oracleConnection);
• Create a GraphOracleSem Object
  - graph = new GraphOracleSem(oracle, model_name, attachment);
• Load data
  - graph.add(Triple.create(...));  // for incremental triple additions
• Collect statistics
  - graph.analyze();
• Run inference
  - graph.performInference();
• Collect statistics
  - graph.analyzeInferredGraph();
• Query
  - QueryFactory.create(...);
  - queryExec = QueryExecutionFactory.create(query, model);
  - resultSet = queryExec.execSelect();
Jena Adaptor for Oracle Database 11g Release 2 (1)

- SPARQL service endpoint supporting full SPARQL Protocol
  - Integrated with Jena Joseki 3.4.0 (deployed in WLS 10.3)
  - Utilizing J2EE data source for DB connection specification
  - SPARQL Update (SPARUL) supported

- Tight integration with Jena ARQ 2.7.0 (2.8.0) for faster query performance
  - Translate OPTIONAL, UNION, FILTER based queries into a single SEM_MATCH
  - Translate Property Path queries into plain SQL
  - Translate conjunctive pattern based queries into plain SQL
  - Translate BGP + parallel OPTIONALS (no nesting) into plain SQL (result_cache enabled)
  - Fall back to the original Jena ARQ in case of failure
Query management and execution control
- Timeout
- Query abort framework
  - Including monitoring threads and a management servlet
  - Designed for a J2EE cluster environment
- Hints allowed in SPARQL query syntax
- Parallel execution

Support ARQ functions for projected variables
- fn:lower-case, upper-case, substring, …

Native, system provided functions can be used in SPARQL
- oext:lower-literal, oext:upper-literal, oext:build-uri-for-id, …
Jena Adaptor for Oracle Database 11g Release 2 (3)

- Extensible user-defined functions in SPARQL
  - Example
    ```sparql
    PREFIX ouext: <http://oracle.com/semtech/jena-adaptor/ext/user-def-function#>
    SELECT ?subject ?object (ouext:my_strlen(?object) as ?obj1)
    WHERE { ?subject dc:title ?object }
    ```

  - User can implement the `my_strlen` functions in Oracle Database

- Connection Pooling through OraclePool
  ```java
  java.util.Properties prop = new java.util.Properties();
  prop.setProperty("InitialLimit", "2"); // create 2 connections
  prop.setProperty("InactivityTimeout", "1800"); // seconds
  ...
  OraclePool op = new OraclePool(szJdbcURL, szUser, szPasswd, prop,
  "OracleSemConnPool");
  Oracle oracle = op.getOracle();
  ```
Jena Adaptor for Oracle Database 11g Release 2 (4)

- API enhancements
  - Parallel statistics collection
  - Parallel and incremental inference
  - Parallel application table index building
  - Query just inferred data
  - Transparent Virtual model support
  - Shutdown hook logic disabled
  - Basic compression enabled by default for application tables
  - OracleUserRule now accepts filter parameter
  - More utility functions…
Semantic Technologies Utilities for Oracle Database 11g Release 2
Utility APIs

- **SEM_APIS.remove_duplicates**
  - e.g. `exec sem_apis.remove_duplicates('graph_model');`

- **SEM_APIS.merge_models**
  - Can be used to clone model as well.
  - e.g. `exec sem_apis.merge_models('model1','model2');`

- **SEM_APIS.swap_names**
  - e.g. `exec sem_apis.swap_names('production_model','prototype_model');`

- **SEM_APIS.alter_model** (entailment)
  - e.g. `sem_apis.alter_model('m1', 'MOVE', 'TBS_SLOWER');`

- **SEM_APIS.rename_model/rename_entailment**
Performance and Scalability Evaluation
Setup for Performance

• Use a balanced hardware system for database
  – A single, huge physical disk for everything is **not** recommended.
  – Multiple hard disks tied together through ASM is a good practice
  – Make sure throughput of hardware components **match** up

<table>
<thead>
<tr>
<th>Component</th>
<th>Hardware spec</th>
<th>Sustained throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU core</td>
<td>-</td>
<td>100 - 200 MB/s</td>
</tr>
<tr>
<td>1/2 Gbit HBA</td>
<td>1/2 Gbit/s</td>
<td>100/200 MB/s</td>
</tr>
<tr>
<td>16 port switch</td>
<td>8 * 2 Gbit/s</td>
<td>1,200 MB/s</td>
</tr>
<tr>
<td>Fiber channel</td>
<td>2 Gbit/s</td>
<td>200 MB/s</td>
</tr>
<tr>
<td>Disk controller</td>
<td>2 Gbit/s</td>
<td>200 MB/s</td>
</tr>
<tr>
<td>GigE NIC (interconnect)</td>
<td>2 Gbit/s</td>
<td>80 MB/s*</td>
</tr>
<tr>
<td>Disk (spindle)</td>
<td>2 Gbit/s</td>
<td>30 - 50 MB/s</td>
</tr>
<tr>
<td>MEM</td>
<td></td>
<td>2k-7k MB/s</td>
</tr>
</tbody>
</table>

Some numbers are from Data Warehousing with 11g and RAC presentation
More tips can be found at http://www.oracle.com/technology/tech/semantic_technologies/pdf/semtech09.pdf
# Bulk Loader Performance on Desktop PC

<table>
<thead>
<tr>
<th>Ontology size</th>
<th>Time</th>
<th>Space (in GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>bulk-load API(^{[1]}) Time</td>
<td>Sql*loader time range low(^{[2]}) high(^{[3]})</td>
</tr>
<tr>
<td>LUBM50 6.9 million</td>
<td>8 min</td>
<td>1min 4.3min</td>
</tr>
<tr>
<td>LUBM1000 138 million</td>
<td>3hr 25min</td>
<td>19min 1h 26m</td>
</tr>
<tr>
<td>LUBM8000 1,106 million</td>
<td>30hr 43min</td>
<td>2h 35m 11h 32m</td>
</tr>
<tr>
<td>UniProt (old) 207 million</td>
<td>4hr 40min</td>
<td>30m 1h 55m</td>
</tr>
</tbody>
</table>

- Results collected on a single CPU PC (3GHz), 4GB RAM, 7200rpm SATA 3.0Gbps, 32 bit Linux. RDBMS 11.1.0.6
- Empty network is assumed

\(^{[1]}\) Uses flags=>' VALUES_TABLE_INDEX_REBUILD ' \(^{[2]}\) Less time for minimal syntax check. \(^{[3]}\) More time is needed when RDF values used in N-Triple file are checked for correctness. \(^{[4]}\) Application table has table compression enabled.\(^{[5]}\) Staging table has table compression enabled.
## Query Performance

<table>
<thead>
<tr>
<th>Ontology LUBM50 6.8 million &amp; 5.4 million inferred</th>
<th>LUBM Benchmark Queries</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Q1</td>
</tr>
<tr>
<td><strong># answers</strong></td>
<td>4</td>
</tr>
<tr>
<td><strong>Complete?</strong></td>
<td>Y</td>
</tr>
<tr>
<td><strong>Time (sec)</strong></td>
<td>0.05</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OWLPrime &amp; new inference components</th>
<th>Query</th>
<th>Q8</th>
<th>Q9</th>
<th>Q10</th>
<th>Q11</th>
<th>Q12</th>
<th>Q13</th>
<th>Q14</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong># answers</strong></td>
<td>7790</td>
<td>13639</td>
<td>4</td>
<td>224</td>
<td>15</td>
<td>228</td>
<td>393730</td>
<td></td>
</tr>
<tr>
<td><strong>Complete?</strong></td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td></td>
</tr>
<tr>
<td><strong>Time (sec)</strong></td>
<td>1.07</td>
<td>1.65</td>
<td>0.01</td>
<td>0.02</td>
<td>0.03</td>
<td>0.01</td>
<td>1.47</td>
<td></td>
</tr>
</tbody>
</table>

- Setup: Intel Q6600 quad-core, 3 7200RPM SATA disks, 8GB DDR2 PC6400 RAM, No RAID. 64-bit Linux 2.6.18. Average of 3 warm runs
Query Performance - Running in Parallel

LUBM1000 Query Performance

- Setup: Server class machine with 16 cores, NAND based flash storage, 32GB RAM, Linux 64 bit, Average of 3 warm runs
11.1.0.7 Inference Performance

- OWLPrime (11.1.0.7) inference performance scales really well with hardware. It is not a parallel inference engine though.
## 11.2 Inference Performance

<table>
<thead>
<tr>
<th>Type</th>
<th>Dataset Details</th>
<th>Time to finish inference:</th>
<th>Comparison</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Parallel Inference</strong></td>
<td>(LUBM8000 1.06 billion triples + 860M inferred)</td>
<td>12 hrs.</td>
<td>3.3x faster compared to serial inference in release 11.1</td>
</tr>
<tr>
<td></td>
<td>(LUBM25000 3.3 billion triples + 2.7 billion inferred)</td>
<td>40 hrs.</td>
<td>30% faster than nearest competitor</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1/5 cost of other hardware configurations</td>
</tr>
<tr>
<td><strong>Incremental Inference</strong></td>
<td>(LUBM8000 1.06 billion triples + 860M inferred)</td>
<td>less than 30 seconds</td>
<td>At least 15x to 50x faster than a complete inference done with release 11.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>after adding 100 triples.</td>
<td></td>
</tr>
<tr>
<td><strong>Large scale owl:sameAs Inference</strong></td>
<td>(UniProt 1 Million sample)</td>
<td></td>
<td>60% less disk space required</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>10x faster inference compared to release 11.1</td>
</tr>
</tbody>
</table>

- **Setup:** Intel Q6600 quad-core, 3 7200RPM SATA disks, 8GB DDR2 PC6400 RAM, No RAID.
  64-bit Linux 2.6.18.  **Assembly cost: less than USD 1,000**
Summary

- Requirement to search and derive greater business knowledge from enterprise databases and applications
- Oracle database 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 (+10 billion)
- Choice of SQL/PLSQL APIs or Java APIs
- Leverages Oracle Partitioning and Compression. RAC supported
- Growing ecosystem of 3rd party tools partners
For More Information

search.oracle.com

Semantic Technologies

or

oracle.com
ORACLE IS THE INFORMATION COMPANY