# RDF Data Management

Amol Deshpande

CMSC 724

# Outline

- Linked Data Introduction
  - From: http://linkeddata.org/guides-and-tutorials
- SPARQL
- RDF Data Management
- SW-Store

# SPARQL By Example

## A Tutorial

*Lee Feigenbaum*
*VP Technology & Standards, Cambridge Semantics*
*Co-Chair, W3C SPARQL Working Group*
*Eric Prud'hommeaux*
*Sanitation Engineer, W3C*
*SPARQL, HCLS, RDB2RDF Working Groups*

---

- Follow along at http://www.cambridgesemantics.com/2008/09/sparql-by-example/.
- Companion "cheat sheet" at http://www.slideshare.net/LeeFeigenbaum/sparql-cheat-sheet.
- Last modified: 2011-01-25
- 


# SPARQL Architecture & Endpoints

- SPARQL queries are executed against *RDF datasets*, consisting of RDF graphs. (More on this later.)
- A *SPARQL endpoint* accepts queries and returns results via HTTP.
  - *Generic endpoints will query any Web-accessible RDF data*
  - *Specific endpoints are hardwired to query against particular datasets*
- The results of SPARQL queries can be returned and/or rendered in a variety of formats:
  - *XML. SPARQL specifies an XML vocabulary for returning tables of results.*
  - *JSON. A JSON "port" of the XML vocabulary, particularly useful for Web applications.*
  - *RDF. Certain SPARQL result clauses trigger RDF responses, which in turn can be serialized in a number of ways (RDF/XML, N-Triples, Turtle, etc.)*
  - *HTML. When using an interactive form to work with SPARQL queries. Often implemented by applying an XSL transform to XML results.*


# Dataset: Friend of a Friend (FOAF)

- FOAF is a standard RDF vocabulary for describing people and relationships
- Tim Berners-Lee's FOAF information available at `http://dig.csail.mit.edu/2008/webdav/timbl/foaf.rdf`
- For our first query, let's find all the names of people mentioned in Tim's FOAF file:

```
@prefix card: <http://www.w3.org/People/Berners-Lee/card#> .
@prefix foaf:  <http://xmlns.com/foaf/0.1/> .

card:i foaf:name
"Timothy Berners-Lee" .
<http://bblfish.net/people/henry/card#me> foaf:name "Henry Story" .
<http://www.cambridgesemantics.com/people/about/lee> foaf:name "Lee Feigenbaum" .
card:amy foaf:name "Amy
            van der Hiel" .

...
```

http://dig.csail.mit.edu/2008/webdav/timbl/foaf.rdf


# Query #1: SELECT, variables, and a triple pattern

*In the graph* **`http://dig.csail.mit.edu/2008/webdav/timbl/foaf.rdf`**, *find me all subjects (*`?person`*) and objects (*`?name`*) linked with the* **`foaf:name`** *predicate. Then return all the values of* `?name`*. In other words, find all names mentioned in Tim Berners-Lee's FOAF file.*

```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
SELECT ?name
WHERE {
    ?person foaf:name ?name .
}
```

- SPARQL *variables* start with a `?` and can match any node (resource or literal) in the RDF dataset.
- *Triple patterns* are just like triples, except that any of the parts of a triple can be replaced with a variable.
- The ***`SELECT`*** result clause returns a table of variables and values that satisfy the query.
- Dataset: **`http://dig.csail.mit.edu/2008/webdav/timbl/foaf.rdf`**


# Query #3: Multiple triple patterns: traversing a graph

*Find me the homepage of anyone known by Tim Berners-Lee.*

```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
PREFIX card: <http://www.w3.org/People/Berners-Lee/card#>
SELECT ?homepage
FROM <http://dig.csail.mit.edu/2008/webdav/timbl/foaf.rdf>
WHERE {
    card:i foaf:knows
?known .
    ?known foaf:homepage ?homepage .
}
```

- The ***FROM*** keyword lets us specify the target graph in the query itself.
- By using **?known** as an object of one triple and the subject of another, we traverse multiple links in the graph.


Try it with ARQ, OpenLink's Virtuoso, or Redland's Rasqal. (Expected results.)


# Query #5: Basic SPARQL filters

*Find me all landlocked countries with a population greater than 15 million.*

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX type: <http://dbpedia.org/class/yago/>
PREFIX prop: <http://dbpedia.org/property/>
SELECT ?country_name ?population
WHERE {
    ?country a type:LandlockedCountries ;
             rdfs:label ?country_name ;
             prop:populationEstimate ?population .

FILTER (?population > 15000000) .
}
```

- ***FILTER*** constraints use boolean conditions to filter out unwanted query results.
- Shortcut: a semicolon (`;`) can be used to separate two triple patterns that share the same subject. (`?country` is the shared subject above.)
- `rdfs:label` is a common predicate for giving a human-friendly label to a resource.
- Note all the translated duplicates in the results. How can we deal with that?

Try it with one of DBPedia's SPARQL endpoint. (Expected results.)

# Query #7b: Finding artists' info - the right way

*Find all Jamendo artists along with their image, home page, and the location they're near, if any.*

```
PREFIX mo: <http://purl.org/ontology/mo/>
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
SELECT ?name ?img ?hp ?loc
WHERE {
  ?a a mo:MusicArtist ;
     foaf:name ?name .

OPTIONAL { ?a foaf:img ?img }
  OPTIONAL { ?a foaf:homepage ?hp }
  OPTIONAL { ?a foaf:based_near ?loc }
}
```

- Not every artist has an image, homepage, or location!
- ***OPTIONAL*** tries to match a graph pattern, but doesn't fail the whole query if the optional match fails.
- If an **OPTIONAL** pattern fails to match for a particular solution, any variables in that pattern remain *unbound* (no value) for that solution.

Try it with DBTune.org's Jamendo-specific SPARQL endpoint. Be sure to choose SPARQL rather than SeRQL. (Expected results.)

# Outline

- Linked Data Introduction
  - From: http://linkeddata.org/guides-and-tutorials
- SPARQL
- RDF Data Management
- SW-Store

# RDF Data Management

- Relational approaches
  - From a survey by Sakr et al.; SIGMOD Record 2010
  - Vertical (triple) table stores: use 3-column tables
    - 3store, RDF-3X, Hexastore…
    - Too many self-joins required
    - Use extensive indexes to speed up queries
  - Property (n-ary) table stores: combine multiple properties
    - Jena1, Jena2, …
    - Issues: NULLs, Multi-valued Attributes, Too many unions and joins
  - Horizontal (binary) table stores: one table for each RDF property
    - SW-Store
    - Issues: Expensive inserts, re-construction costs high

# RDF Data Management

- Non-relational approaches
  - Using a graph database
  - Many graph databases (e.g., AllegroGraph) actually use one of the approaches from previous page
  - Neo4j stores the graph as it is
    - Not sure if it is very good with RDF data
- Overall, much ongoing work on this problem in several communities

# Outline

- Linked Data Introduction
  - From: http://linkeddata.org/guides-and-tutorials
- SPARQL
- RDF Data Management
- SW-Store

| Subj. | Prop. | Obj. |
|---|---|---|
| ID1 | type | BookType |
| ID1 | title | “XYZ” |
| ID1 | author | “Fox, Joe” |
| ID1 | copyright | “2001” |
| ID2 | type | CDType |
| ID2 | title | “ABC” |
| ID2 | artist | “Orr, Tim” |
| ID2 | copyright | “1985” |
| ID2 | language | “French” |
| ID3 | type | BookType |
| ID3 | title | “MNO” |
| ID3 | language | “English” |
| ID4 | type | DVDType |
| ID4 | title | “DEF” |
| ID5 | type | CDType |
| ID5 | title | “GHI” |
| ID5 | copyright | “1995” |
| ID6 | type | BookType |
| ID6 | copyright | “2004” |

**(a) Some Example RDF Triples**

```
SELECT C.obj.
FROM TRIPLES AS A,
     TRIPLES AS B,
     TRIPLES AS C
WHERE A.subj. = B.subj.
  AND B.subj. = C.subj.
  AND A.prop. = ‘copyright’
  AND A.obj. = ‘‘2001’’
  AND B.prop. = ‘author’
  AND B.obj. = ‘‘Fox, Joe’’
  AND C.prop. = ‘title’
```

**(b) Example SQL Query Over RDF Triples Table From (a)**

**Property Table**

| Subj. | Type | Title | copyright |
|---|---|---|---|
| ID1 | BookType | “XYZ” | “2001” |
| ID2 | CDType | “ABC” | “1985” |
| ID3 | BookType | “MNP” | NULL |
| ID4 | DVDType | “DEF” | NULL |
| ID5 | CDType | “GHI” | “1995” |
| ID6 | BookType | NULL | “2004” |

**Left-Over Triples**

| Subj. | Prop. | Obj. |
|---|---|---|
| ID1 | author | “Fox, Joe” |
| ID2 | artist | “Orr, Tim” |
| ID2 | language | “French” |
| ID3 | language | “English” |

**(c) Clustered Property Table Example**

**Class: BookType**

| Subj. | Title | Author | copyright |
|---|---|---|---|
| ID1 | “XYZ” | “Fox, Joe” | “2001” |
| ID3 | “MNP” | NULL | NULL |
| ID6 | NULL | NULL | “2004” |

**Class: CDType**

| Subj. | Title | Artist | copyright |
|---|---|---|---|
| ID2 | “ABC” | “Orr, Tim” | “1985” |
| ID5 | “GHI” | NULL | “1995” |

**Left-Over Triples**

| Subj. | Prop. | Obj. |
|---|---|---|
| ID2 | language | “French” |
| ID3 | language | “English” |
| ID4 | type | DVDType |
| ID4 | title | “DEF” |

**(d) Property-Class Table Example**

**Table 1: Some sample RDF data and possible property tables.**

# SW-Store Overview

- One table for each property
  - No NULLs; multi-valued attributes not a problem
- Use C-Store for storing this
  - Lower overheads for storing narrow tables
  - Joins can be done using specialized merge routines
- Extensive experiments indicating huge benefits

**Type**

| | |
|---|---|
| ID1 | BookType |
| ID2 | CDType |
| ID3 | BookType |
| ID4 | DVDType |
| ID5 | CDType |
| ID6 | BookType |

**Author**

| | |
|---|---|
| ID1 | "Fox, Joe" |

**Title**

| | |
|---|---|
| ID1 | "XYZ" |
| ID2 | "ABC" |
| ID3 | "MNO" |
| ID4 | "DEF" |
| ID5 | "GHI" |

**Artist**

| | |
|---|---|
| ID2 | "Orr, Tim" |

**Copyright**

| | |
|---|---|
| ID1 | "2001" |
| ID2 | "1985" |
| ID5 | "1995" |
| ID6 | "2004" |

**Language**

| | |
|---|---|
| ID2 | "French" |
| ID3 | "English" |