Querying RDF, RDFS, OWL

Partially adapted from Lee Feigenbaum and Olaf Hartig’s slides
What is a Graph Query Language?

- A Graph Query language should allow us to
  - Retrieve any query-specified portion of some graph data
  - Create a new graph by combining different pieces of retrieved subgraphs in a query-specified way
  - Compute a set of graph properties
    - Diameter
    - Distance between two nodes
    - Centrality of nodes
    - ...

- We will discuss SPARQL
  - Standard RDF Query Language
  - SPARQL only allows us to do a few of the operations an ideal graph query language should
A Single Variable Graph Pattern

</trouble_with_bob>

blog:comment

dc:title

dc:creator

"Trouble with Bob"

<http://eve/>

foaf:interest

http://xtech2008.org

<http://bob/alice_rules>

blog:comment

dc:title

dc:creator

"Alice rules"

<http://bob/>

foaf:interest

http://www2008.org

<http://eve/> foaf:interest ?x

http://xtech.2008.org
SELECT ?x, ?y where {?x foaf:interest ?y}

<table>
<thead>
<tr>
<th>?x</th>
<th>?y</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://eve/">http://eve/</a></td>
<td><a href="http://xtech.org">http://xtech.org</a></td>
</tr>
</tbody>
</table>
Basic Graph Patterns

- What has Alice written?
  - BGP
  
  ```turtle
  {?x dc:creator http://alice/ .
   ?x dc:title ?y}
  ```

- Who has common interests?
  - BGP
  
  ```turtle
  {?x foaf:interest ?y .
   ?z foaf:interest ?y}
  ```

- Matching Literals
  - Consider the data
    - Will it match
      - `{?x hasPet "cat"}` ?
      - `{?x hasAge 29}` ?

  ```turtle
  http://alice/ hasAge 29^^xsd:integer
  http://alice/ hasPet "cat"@en
  ```
Structure of a SPARQL Query

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT ?x, ?z, ?y
FROM <http://example.org/Hackers>
WHERE {
  {?x dc:creator ?z .
   ?x dc:title ?y}
}
ORDER BY ?y

- **Prologue:**
  - Prefix definitions are references in the query
  - No period ("." ) character to separate (as in N3)
  - If we said **PREFIX** : [http://example.org/Hackers](http://example.org/Hackers)
    - We could drop the FROM clause
    - We have to say :?x etc.
# Structure of a SPARQL Query

**Result form specification:**

- **SELECT, DESCRIBE, CONSTRUCT, or ASK**
- **SELECT:** - Variable list or asterisk ("*") character for all
  - DISTINCT for disjoint results
Dataset specification:

- Specify the datasets to be queried
- FROM and FROM NAMED clauses (each with a URI)
- When multiple datasets are specified, the system assumes an RDF merge of the two graphs
- FROM NAMED is discussed later
Structure of a SPARQL Query

- **Solution modifiers:**
  - Modify the result set, but not the single results
  - ORDER BY, LIMIT, or OFFSET
    - **LIMIT** gets a query-specified number of results
    - **OFFSET** $k$ gets results starting from the $k$-th result record

```sparql
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT ?x, ?z, ?y
FROM <http://example.org/Hackers>
WHERE {
  {?x dc:creator ?z .
   ?x dc:title ?y}
}
ORDER BY ?y
```
Graph Patterns in SPARQL

- Basic graph pattern (BGP)
- Optional graph pattern
- Union graph pattern
- (Constraints)
- Graph graph pattern
- Group graph pattern
More on BGPs

- **Using Blank Nodes in Queries**
  - Blank nodes in graph patterns *act as variables, not as references to specific blank nodes in the data being queried.*
  - Permitted as subject and object of a triple pattern
    - Non-selectable variables
    - Indicated either as `_`:abc or as `[ ]`

- Blank node identifiers can appear in query results

```
_:b50 dc:creator ?x.
_:b50 dc:title ?title


?x blog:comment _:b57.


?x blog_comment [ dc:title ?title ]
```
Optional Graph Patterns

- Who commented on “trouble_with_bob”?

select ?p, ?t
where {Trouble_with_bob blog:comment ?y .
?y dc:title ?t}

- Does not report eve

select ?p, ?t
where {Trouble_with_bob blog:comment ?y .
optional {?y dc:title ?t}}

- Reports eve
Union Graph Patterns

- Who is interested in the conferences Xtech 2008 or WWW 2008?

```sparql
select ?x
where {
  {?x foaf:interest http://xTech2008/}
  UNION
  {?x foaf:interest http://www2008/}
}
```

- Union patterns are used to query for alternatives

```sparql
select ?x, ?y
where {
  {John foaf:interest ?x }
  UNION
  {John likes ?y}
}
```
Constraints – Filters

- Constraints filter solutions
  - Keyword **FILTER** followed by expression
  - Filter expressions contain operators and functions

```sparql
select ?y
where {
  _b20 dc:title ?y
  filter regex(?y "rule")
}
```

<table>
<thead>
<tr>
<th>?y</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice Rules</td>
</tr>
</tbody>
</table>

### Built-in Constraints

- **Unary Operators**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Type(A)</th>
<th>Result type</th>
</tr>
</thead>
<tbody>
<tr>
<td>! A</td>
<td>xsd:boolean</td>
<td>xsd:boolean</td>
</tr>
<tr>
<td>+ A</td>
<td>numeric</td>
<td>numeric</td>
</tr>
<tr>
<td>- A</td>
<td>numeric</td>
<td>numeric</td>
</tr>
<tr>
<td>BOUND(A)</td>
<td>variable</td>
<td>xsd:boolean</td>
</tr>
<tr>
<td>isURI(A)</td>
<td>RDF term</td>
<td>xsd:boolean</td>
</tr>
<tr>
<td>isBLANK(A)</td>
<td>RDF term</td>
<td>xsd:boolean</td>
</tr>
<tr>
<td>isLITERAL(A)</td>
<td>RDF term</td>
<td>xsd:boolean</td>
</tr>
<tr>
<td>STR(A)</td>
<td>literal / URI</td>
<td>simple literal</td>
</tr>
<tr>
<td>LANG(A)</td>
<td>literal</td>
<td>simple literal</td>
</tr>
<tr>
<td>DATATYPE(A)</td>
<td>literal</td>
<td>simple literal</td>
</tr>
</tbody>
</table>
Filter Example

- Find me all landlocked countries with a population greater than 15 million with the highest population country first.

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 && langMatches(lang(?country_name), "EN")) .
} ORDER BY DESC(?population)

- Try this at http://dbpedia.org/sparql
Homework from DBPedia

- Find everything about the country whose name is Afghanistan in language English
  - everything means all properties of the country
- Who is Barak Obama?
- Where is Greece?
- What is the capital of Nepal?
- What is the area of work of Albert Einstein?
- How is India related to “Indira Gandhi”?
Group Graph Patterns

- Consider the query

```sparql
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 &&
    langMatches(lang(?country_name), "EN"))
  }
  {
    ?place prop:establishedDate ?y .
    FILTER (?y > 1980)
  }.
FILTER (?country = ?place)
} ORDER BY DESC(?population)
```

- Groups break up a graph pattern into multiple pieces such that filters can be applied to each piece and joint filters can be applied across groups
Find cities in the UK whose name is not Manchester.

PREFIX prop: <http://dbpedia.org/property/>
SELECT DISTINCT ?x WHERE {
  ?x a <http://schema.org/City>.
  FILTER (str(?city) != "Manchester").
  ?x prop:subdivisionName ?y.
  FILTER(str(?y) = "United Kingdom").
} ORDER BY desc(?x)
Negation with SPARQL Filters

<table>
<thead>
<tr>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://dbpedia.org/resource/Stoke-on-Trent">http://dbpedia.org/resource/Stoke-on-Trent</a></td>
</tr>
<tr>
<td><a href="http://dbpedia.org/resource/Sheffield">http://dbpedia.org/resource/Sheffield</a></td>
</tr>
<tr>
<td><a href="http://dbpedia.org/resource/Portsmouth">http://dbpedia.org/resource/Portsmouth</a></td>
</tr>
<tr>
<td><a href="http://dbpedia.org/resource/Plymouth">http://dbpedia.org/resource/Plymouth</a></td>
</tr>
<tr>
<td><a href="http://dbpedia.org/resource/Newcastle_upon_Tyne">http://dbpedia.org/resource/Newcastle_upon_Tyne</a></td>
</tr>
<tr>
<td><a href="http://dbpedia.org/resource/Manchester">http://dbpedia.org/resource/Manchester</a></td>
</tr>
<tr>
<td><a href="http://dbpedia.org/resource/Kingston_upon_Hull">http://dbpedia.org/resource/Kingston_upon_Hull</a></td>
</tr>
<tr>
<td><a href="http://dbpedia.org/resource/Hamilton,_Bermuda">http://dbpedia.org/resource/Hamilton,_Bermuda</a></td>
</tr>
<tr>
<td><a href="http://dbpedia.org/resource/Edinburgh">http://dbpedia.org/resource/Edinburgh</a></td>
</tr>
<tr>
<td><a href="http://dbpedia.org/resource/City_of_Sunderland">http://dbpedia.org/resource/City_of_Sunderland</a></td>
</tr>
<tr>
<td><a href="http://dbpedia.org/resource/City_of_Salford">http://dbpedia.org/resource/City_of_Salford</a></td>
</tr>
<tr>
<td><a href="http://dbpedia.org/resource/City_of_Lancaster">http://dbpedia.org/resource/City_of_Lancaster</a></td>
</tr>
<tr>
<td><a href="http://dbpedia.org/resource/City_of_Carlisle">http://dbpedia.org/resource/City_of_Carlisle</a></td>
</tr>
<tr>
<td><a href="http://dbpedia.org/resource/City_of_Bradford">http://dbpedia.org/resource/City_of_Bradford</a></td>
</tr>
<tr>
<td><a href="http://dbpedia.org/resource/Bristol">http://dbpedia.org/resource/Bristol</a></td>
</tr>
<tr>
<td><a href="http://dbpedia.org/resource/Brades">http://dbpedia.org/resource/Brades</a></td>
</tr>
<tr>
<td><a href="http://dbpedia.org/resource/Birmingham">http://dbpedia.org/resource/Birmingham</a></td>
</tr>
</tbody>
</table>

What is this? Is this result incorrect?
Negation in SPARQL Filters (contd.)

- Desired behavior:
  - **Negation by Failure**

  - **Negation as failure** is a non-monotonic inference rule in logic programming, used to derive predicate \( \text{not}(p) \) from failure to derive predicate \( p \)

  - First try to satisfy the predicate \( p \), and test if you failed. If you did, declare the result as satisfying \( \text{not}(p) \)

```sparql
PREFIX prop: <http://dbpedia.org/property/>

SELECT distinct ?x
WHERE {
  ?x a <http://schema.org/City>.
  ?x prop:subdivisionName ?y.
  FILTER(str(?y) = "United Kingdom").
  OPTIONAL{?x rdfs:label ?city.
    FILTER (str(?city) = "Manchester")}.
  FILTER(!bound(?city))
} ORDER BY desc(?x)
```

A logic exercise

\[
p \leftarrow q \land \text{not } r \\
q \leftarrow s \\
q \leftarrow t \\
t \leftarrow
\]

is \( p \) true?
A *merge* of a set of RDF graphs is defined as follows.

- If the graphs in the set have *no blank nodes* in common, then the union of the graphs is a merge.
- If they *do share blank nodes*, then it is the union of a set of graphs that is obtained by replacing the graphs in the set by equivalent graphs that share no blank nodes. This is often described by saying that the blank nodes have been 'standardized apart'.

- Using the convention on equivalent graphs and identity, any graph in the original set is considered to be a subgraph of the merge.

One does not obtain the merge of a set of graphs by concatenating their corresponding n-Triples documents and constructing the graph described by the merged document.

- If some of the documents use the same node identifiers, the merged document will describe a graph in which some of the blank nodes have been 'accidentally' identified.
- To merge n-Triples documents it is necessary to check if the same nodeID is used in two or more documents, and to replace it with a distinct nodeID in each of them, before merging the documents.
SPARQL queries are executed against an RDF dataset

- An RDF dataset comprises:
  - One default graph and
  - Zero or more named graphs (identified by an URI)

Keyword GRAPH makes one of the named graphs the *active graph* used for pattern matching
A Graph that refers to other graphs

- **A base graph** `ds-dft.ttl`

  ```
  @prefix dc: <http://purl.org/dc/elements/1.1/> .
  @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

  ```

- **A query on the base graph**

  ```
  PREFIX xsd: http://www.w3.org/2001/XMLSchema#
  PREFIX dc: <http://purl.org/dc/elements/1.1/>
  PREFIX : <.>
  select * { ?s ?p ?o }
  ```

- **The result**

<table>
<thead>
<tr>
<th>s</th>
<th>p</th>
<th>o</th>
</tr>
</thead>
<tbody>
<tr>
<td>:ds-ng-1.ttl</td>
<td>dc:date</td>
<td>&quot;2005-07-14T03:18:56+01:00&quot;^^xsd:dateTime</td>
</tr>
<tr>
<td>:ds-ng-2.ttl</td>
<td>dc:date</td>
<td>&quot;2005-09-22T05:53:05+0100&quot;^^xsd:dateTime</td>
</tr>
</tbody>
</table>
Adding Named Graphs

- **ds-ng-1.ttl**

  ```ttl
  @prefix dc: <http://purl.org/dc/elements/1.1/> .
  [] dc:title "Harry Potter and the Philosopher's Stone" .
  [] dc:title "Harry Potter and the Chamber of Secrets" .
  ```

- **ds-ng-2.ttl**

  ```ttl
  @prefix dc: <http://purl.org/dc/elements/1.1/> .
  [] dc:title "Harry Potter and the Sorcerer's Stone" .
  [] dc:title "Harry Potter and the Chamber of Secrets" .
  ```
Putting them together

Query

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX : <.>

select * { { ?s ?p ?o } union { graph ?g { ?s ?p ?o } } }
```
## Result of a Graph Pattern Query

<table>
<thead>
<tr>
<th>s</th>
<th>p</th>
<th>o</th>
<th>g</th>
</tr>
</thead>
<tbody>
<tr>
<td>:ds-ng-1.ttl</td>
<td>dc:date</td>
<td>&quot;2005-07-14T03:18:56+01:00&quot;^^xsd:dateTime</td>
<td></td>
</tr>
<tr>
<td>:ds-ng-2.ttl</td>
<td>dc:date</td>
<td>&quot;2005-09-22T05:53:05+0100&quot;^^xsd:dateTime</td>
<td></td>
</tr>
<tr>
<td>_b2</td>
<td>dc:title</td>
<td>&quot;Harry Potter and the Philosopher's Stone&quot;</td>
<td>:ds-ng-1.ttl</td>
</tr>
<tr>
<td>_b3</td>
<td>dc:title</td>
<td>&quot;Harry Potter and the Chamber of Secrets&quot;</td>
<td>:ds-ng-1.ttl</td>
</tr>
<tr>
<td>_b4</td>
<td>dc:title</td>
<td>&quot;Harry Potter and the Sorcerer's Stone&quot;</td>
<td>:ds-ng-2.ttl</td>
</tr>
<tr>
<td>_b5</td>
<td>dc:title</td>
<td>&quot;Harry Potter and the Chamber of Secrets&quot;</td>
<td>:ds-ng-2.ttl</td>
</tr>
</tbody>
</table>

- The **graph** construct in the query instantiates every graph type variable with the content and merges the result graph
  - Recall that the Prefix clause did not include these graphs
Querying Component Graphs

- Querying one graph
  - First finding a matching graph and then retrieving data from it

```prefix
xsd: <http://www.w3.org/2001/XMLSchema#>
dc: <http://purl.org/dc/elements/1.1/>
.: <.>

select ?title
{
  graph :ds-ng-2.ttl { ?s ?p ?o }
}
```

```prefix
xsd: <http://www.w3.org/2001/XMLSchema#>
dc: <http://purl.org/dc/elements/1.1/>
.: <.>

select ?date ?title
{
  ?g dc:date ?date .
  FILTER (?date > "2005-08-01T00:00:00Z"^^xsd:dateTime )
  graph ?g { ?s dc:title ?title }
}
```
The “Named Graph” Construct

- There are many graphs but you want to query only a few of them
- The NAMED GRAPH construct constrains the universe of active graphs that you want to query

```sql
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX : <.>

select *
from <ds-dft.ttl>
from named <ds-ng-1.ttl>
from named <ds-ng-2.ttl>
{
    { ?s ?p ?o } union
    { graph ?g { ?s ?p ?o } }
}
```
Named Graphs

- Finding all named graphs from a data store
  - [http://data.semanticweb.org/snorql/](http://data.semanticweb.org/snorql/)

```sparql
SELECT DISTINCT ?namedgraph ?label
WHERE {
  GRAPH ?namedgraph { ?s ?p ?o }
  OPTIONAL { ?namedgraph rdfs:label ?label }
}
ORDER BY ?namedgraph
```

- What does the following query do?

```sparql
SELECT DISTINCT ?name
WHERE {
  ?person foaf:name ?name
  GRAPH ?g1 { ?person a foaf:Person }
  GRAPH ?g2 { ?person a foaf:Person }
  GRAPH ?g3 { ?person a foaf:Person }
  FILTER(?g1 != ?g2 && ?g1 != ?g3 && ?g2 != ?g3 )
}
```
ASK Queries

- Checks if there is *at least one* result
  - Returns a Boolean response
- No projection variables in an ASK query

```PREFIX prop: <http://dbpedia.org/property/>

ASK {
  FILTER(?amazon > ?nile) .
}
```
DESCRIBE Queries

- Returns an RDF graph with data about resources
- Nondeterministic (i.e. query processor determines the actual structure of the returned RDF graph)
- DESCRIBE ResourceURI is a valid query

PREFIX type: <http://dbpedia.org/class/yago/>
PREFIX prop: <http://dbpedia.org/property/>
DESCRIBE <http://dbpedia.org/resource/George_W._Bush>
A DESCRIBE query can also have projection variables and a WHERE clause

PREFIX type: <http://dbpedia.org/class/yago/>
PREFIX prop: <http://dbpedia.org/property/>
DESCRIBE ?country
WHERE {
    ?country a type:LandlockedCountries;
    rdfs:label ?country_name;
    prop:populationEstimate ?population .
    FILTER (?population > 15000000 && langMatches(lang(?country_name), "EN"))
}
Strategies for DESCRIBE

- Return all triples with *this* resource as subject or object
- Return contents of an authoritative graph for the resource
- Return a *minimum self-contained graph* for the resource
  - The statement in question;
  - Recursively, for all the blank nodes involved by statements included in the description so far, the *MSG* of all the statements involving such blank nodes
- Return *concise bounded descriptions*
  - Include in the subgraph all statements in the source graph where the subject of the statement is the starting node;
  - Recursively, for all statements identified in the subgraph thus far having a blank node object, include in the subgraph all statements in the source graph where the subject of the statement is the blank node in question and which are not already included in the subgraph.
  - Recursively, for all statements included in the subgraph thus far, for all reifications of each statement in the source graph, include the *CBD* beginning from the rdf:Statement node of each reification.
CONSTRUCT Queries

- Returns an RDF graph created from a template
  - Template: graph pattern with variables from the query pattern

```
PREFIX vCard: <http://www.w3.org/2001/vcard-rdf/3.0#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

CONSTRUCT {
}
FROM <http://dig.csail.mit.edu/2008/webdav/timbl/foaf.rdf>
WHERE {
  OPTIONAL { ?X foaf:name ?name . FILTER isLiteral(?name) . }
  OPTIONAL { ?X foaf:homepage ?url . FILTER isURI(?url) . }
  OPTIONAL { ?X foaf:title ?title . FILTER isLiteral(?title) . }
}
```

Triples are not created in the result graph for template patterns that involve an unbound variable.
SPARQL 1.1 – the Next Generation

### Additions to the Query Language:

- Project Expressions
  
  ```sparql
  SELECT ?Item (?Pr * 1.1 AS ?NewP)
  ```

- Aggregate functions
  
  ```sparql
  SELECT (Count(DISTINCT ?T) AS ?C)
  ```

- Subqueries
  
  ```sparql
  CONSTRUCT{ ?P foaf:name ?FullName
  WHERE {
    WHERE { ?P foaf:firstName ?F ; foaf:lastName ?L. }
  }
  }
  WHERE{ ?X rdf:type foaf:Person
  MINUS { ?X foaf:homepage ?H } }
  ```

- Negation
  
  ```sparql
  SELECT DISTINCT ?N
  ```

- Property Paths

- Entailment
  
  ```sparql
  SELECT DISTINCT ?beer
  WHERE {?beer rdf:type/rdfs:subClassOf* beer:Beer}
  ```

- Undetermined so far
General Idea: Answer Queries with implicit answers

Useful for an ontology graph:

**T-Box:**

\[
\text{foaf:Person rdfs:subClassOf foaf:Agent .}
\]
\[
\text{foaf:Person rdfs:subClassOf}
\]
\[
[\text{a owl:Restriction ;}
\]
\[
\text{owl:onProperty :hasFather ;}
\]
\[
\text{owl:someValuesFrom foaf:Person ] .}
\]
\[
\text{foaf:knows rdfs:range foaf:Person .}
\]

**A-Box:**

\[
:jeff \text{ a Person .}
\]
\[
:jeff \text{ foaf:knows :aidan .}
\]

**Query:**

```
SELECT ?X { ?X a foaf:Person }
```

Pure SPARQL 1.0 returns only :Jeff, should also return: aidan
## Some Public SPARQL Endpoints

<table>
<thead>
<tr>
<th>Name</th>
<th>URL</th>
<th>What’s there?</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPARQLer</td>
<td><a href="http://sparql.org/sparql.html">http://sparql.org/sparql.html</a></td>
<td>General-purpose query endpoint for Web-accessible data</td>
</tr>
<tr>
<td>DBLP</td>
<td><a href="http://www4.wiwiss.fu-berlin.de/dblp/snorql/">http://www4.wiwiss.fu-berlin.de/dblp/snorql/</a></td>
<td>Bibliographic data from computer science journals and conferences</td>
</tr>
<tr>
<td>LinkedMDB</td>
<td><a href="http://data.linkedmdb.org/sparql">http://data.linkedmdb.org/sparql</a></td>
<td>Films, actors, directors, writers, producers, etc.</td>
</tr>
<tr>
<td>bio2rdf</td>
<td><a href="http://bio2rdf.org/sparql">http://bio2rdf.org/sparql</a></td>
<td>Bioinformatics data from around 40 public databases</td>
</tr>
</tbody>
</table>