Querying the Semantic Web

CSE 595 – Semantic Web
Stony Brook University

http://www.cs.stonybrook.edu/~cse595
Lecture Outline

- SPARQL Infrastructure
- Basics: Matching Patterns
- Filters
- Constructs for Dealing with an Open World
- Organizing Result Sets
- Other Forms of SPARQL Queries
- Querying Schemas
- Adding Information with SPARQL Update
Why an RDF Query Language?

- SPARQL is specifically designed for RDF, and is tailored to and relies upon the various technologies underlying the web.
  - If you are familiar with database query languages like SQL, you will notice many similarities.
- XML is at a lower level of abstraction than RDF.
  - Thus we would require:
    - XML namespaces
    - several XPath queries
    - XSD data types
SPARQL Infrastructure

• A *triple store* is a database for RDF
  • Within the specifications for SPARQL a triple store is referred to as a *Graph Store*.

• Before one can query a triple store, it needs to be populated with RDF
  • A mechanism called *SPARQL Update* provides a series of options for inserting, loading, and deleting RDF into a triple store

• Most triple stores provide bulk upload options

• Once data is loaded into a triple store, it can be queried by sending SPARQL queries using the SPARQL protocol
SPARQL Infrastructure

- Each triple store provides what is termed an **endpoint**, where SPARQL queries can be submitted.

- Clients send queries to an endpoint using the **HTTP protocol**.
  - Clients can issue a SPARQL query to an endpoint by entering it into the browser’s URL.
  - Better clients designed specifically for SPARQL are used.
  - APIs are also used (e.g., Jena ARQ).
SPARQL Infrastructure

- There are numerous SPARQL endpoints on the web
  - access to large amounts of data
    - For example, [http://dbpedia.org/sparql](http://dbpedia.org/sparql) provides a query endpoint to query over an RDF representation of Wikipedia
    - [https://query.wikidata.org/](https://query.wikidata.org/)
    - [http://babelnet.org/sparql/](http://babelnet.org/sparql/)
  - list of SPARQL endpoints at [http://CKAN.org](http://CKAN.org)
SPARQL Basic Queries

- SPARQL is based on matching **graph patterns**:
  - The simplest graph pattern is the **triple pattern** like an RDF triple, but with the possibility of a variable instead of an RDF term in the subject, predicate, or object positions
    - A variable starts with `?`
  - Combining triple patterns gives a basic graph pattern, where an exact match to a graph is needed to fulfill a pattern
Using select-from-where

- As in SQL, SPARQL queries have a SELECT-FROM-WHERE structure:
  - SELECT specifies the projection: the number and order of retrieved data
  - FROM is used to specify the source being queried (optional)
  - WHERE imposes constraints on possible solutions in the form of graph pattern templates and boolean constraints
- Retrieve all phone numbers of staff members:
  ```sparql
  SELECT ?x ?y
  WHERE
  { ?x uni:phone ?y .}
  ```
  - Here ?x and ?y are variables, and ?x uni:phone ?y represents a resource-property-value triple pattern
Consider the RDF describing the Baron Way apartment and its location:

```xml
@prefix swp: <http://www.semanticwebprimer.org/ontology/apartments.ttl#>.
@prefix dbpedia: <http://dbpedia.org/resource/>.
@prefix dbpedia-owl: <http://dbpedia.org/ontology/>.

swp:BaronWayApartment swp:hasNumberOfBedrooms 3;
    swp:isPartOf swp:BaronWayBuilding.
swp:BaronWayBuilding dbpedia-owl:location
dbpedia:Amsterdam,
dbpedia:Netherlands.
```
Example

- To find the location of the building, a triple pattern is: `swp:BaronWayBuilding dbpedia-owl:location dbpedia:Amsterdam`.
- In SPARQL, we can just replace any element of the triple with a variable: `swp:BaronWayBuilding dbpedia-owl:location ?location`
- The triple store will take this graph pattern and try to find sets of triples that match the pattern:
  - it would return `dbpedia:Amsterdam` and `dbpedia:Netherlands`
  - it finds all triples where `swp:BaronWayBuilding` is in the subject position and `dbpedia-owl:location` is in the predicate position
Example

- A complete SPARQL query also contains all prefixes and we need to tell the triple store that we are interested in the results for a particular variable:

```sparql
PREFIX swp: <http://www.semanticwebprimer.org/ontology/apartments.ttl#>.

SELECT ?location
WHERE {
    swp:BaronWayBuilding dbpedia-owl:location ?location.
}
```
Example

- The results of the query are returned in a set of mappings called *bindings* that denote which elements correspond to a given variable:

```
?location

http://dbpedia.org/resource/Amsterdam.
```
Example

• Find where the BaronWayApartment is located:

```
PREFIX swp: <http://www.semanticwebprimer.org/ontology/apartments.ttl#>.

SELECT ?location
WHERE {
  swp:BaronWayApartment swp:isPartOf ?building.
}
```

• The variable `?building` is in the subject position: variables can occur in any position in the SPARQL query.

• The query reuses the variable name `?building`: find triples where the object of the first statement is the same as the subject of the second statement.
Example

- Find all the information about Baron Way Apartment in the triple store:

```
PREFIX swp: <http://www.semanticwebprimer.org/ontology/apartments.ttl#>.

SELECT ?p ?o
WHERE {
}
```

<table>
<thead>
<tr>
<th>?p</th>
<th>?o</th>
</tr>
</thead>
<tbody>
<tr>
<td>swp:hasNumberOfBedrooms</td>
<td>3</td>
</tr>
<tr>
<td>swp:isPartOf</td>
<td>swp:BaronWayBuilding</td>
</tr>
</tbody>
</table>
Example

• On larger data sets we may not know how many results there are or if our query would return a whole dataset
  • it is fairly easy to write queries that can return millions of triples

PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?c
WHERE
{
  ?c rdf:type rdfs:Class .
}

• Retrieves all triple patterns, where:
  • the property is **rdf:type**
  • the object is **rdfs:Class**
• Which means that it retrieves all classes
Example

- It is good practice to limit the number of answers a query returns, especially when using public endpoints with the **LIMIT** keyword

```prefix
PREFIX swp: <http://www.semanticwebprimer.org/ontology/apartments.ttl#>.
```

```sql
SELECT ?p ?o
WHERE {
}
LIMIT 10
```
Example

- SPARQL provides a way of expressing concisely chains of properties
- instead of:

```sparql
SELECT ?apartment
WHERE {
}
```

we can do:

```sparql
SELECT ?apartment
WHERE {
}
```
Filters

• Find all the apartments that have more than 2 bedrooms:

```sparql
PREFIX swp: <http://www.semanticwebprimer.org/ontology/apartments.ttl#>. 
SELECT ?apartment
WHERE {
    ?apartment swp:hasNumberOfBedrooms ?bedrooms. 
    FILTER (?bedrooms > 2).
}
```

• The syntactic shortcuts for SPARQL and Turtle are the same
  • like Turtle, SPARQL allows for shortened forms of common literals
  • in this case, 2 is a shortcut for "2"^^xsd:integer

• Less than, greater than, and equality are supported for numeric data types (i.e., integers, decimals) as well as date/time
Filters

- Regular expressions for strings
- Assume that our data set contains the triple:
  \texttt{swp:BaronWayApartment swp:address "4 Baron Way Circle"}
- We might like to find all the resources that contain "Baron Way" in their address

\texttt{PREFIX swp: <http://www.semanticwebprimer.org/ontology/apartments.ttl#>.
SELECT ?apartment
WHERE {
  ?apartment swp:address ?address.
  FILTER regex(?address, "Baron Way").
}
Filters

- **regex** is a filter function
- **str** function converts resources and literals into string representations that can then be used in **regex**
  - For example, we can search for "Baron" in the URL of the resource instead of using the label

```sparql
PREFIX swp: <http://www.semanticwebprimer.org/ontology/apartments.ttl#>. 
SELECT ?apartment
WHERE {
    ?apartment swp:address ?address.
    FILTER regex(str(?apartment), "Baron"). 
}
```
- Note that the apartment resource is a URL
Implicit Join

- Retrieve all lecturers and their phone numbers:

  ```sql
  SELECT ?x ?y
  WHERE
  { ?x rdf:type uni:Lecturer ;
    uni:phone ?y . }
  ```

- Implicit join: We restrict the second pattern only to those triples, the resource of which is in the variable `?x`

- Here we use a syntax shortcut as well: a semicolon indicates that the following triple shares its subject with the previous one
Implicit join

- The previous query is equivalent to writing:

```sql
SELECT ?x ?y
WHERE
{
  ?x rdf:type uni:Lecturer .
  ?x uni:phone ?y .
}
```
Explicit Join

- Retrieve the name of all courses taught by the lecturer with ID 949352:

```
SELECT ?n
WHERE
{
    ?x rdf:type uni:Course ;
    uni:isTaughtBy :949352 .
    ?c uni:name ?n .
    FILTER (?c = ?x) .
}
```
Constructs for Dealing with an Open World

- Unlike a traditional database, not every resource on the Semantic Web will be described using the same schema or have all of the same properties
- This is called the open world assumption
  - some apartments may be more well described than others
  - some may be described using a different vocabulary

```turtle
@prefix swp: <http://www.semanticwebprimer.org/ontology/apartments.ttl#>.
@prefix dbpedia: <http://dbpedia.org/resource/>.
@prefix dbpedia-owl: <http://dbpedia.org/ontology/>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
swp:BaronWayApartment swp:hasNumberOfBedrooms 3.
swp:BaronWayApartment dbpedia-owl:location dbpedia:Amsterdam.
swp:BaronWayApartment refs:label "Baron Way Apartment for Rent".
swp:FloridaAveStudio swp:hasNumberOfBedrooms 1.
swp:FloridaAveStudio dbpedia-owl:locationCity dbpedia:Amsterdam.
```
Some results are OPTIONAL:

PREFIX swp: <http://www.semanticwebprimer.org/ontology/apartments.ttl#>.

SELECT ?apartment
WHERE {
  {?apartment dbpedia-owl:location dbpedia:Amsterdam.} 
  UNION 
  {?apartment dbpedia-owl:locationCity dbpedia:Amsterdam.} 
  OPTIONAL 
  {?apartment rdfs:label ?label.}
}

<table>
<thead>
<tr>
<th>?apartment</th>
<th>?label</th>
</tr>
</thead>
<tbody>
<tr>
<td>swp:BaronWayApartment</td>
<td>Baron Way Apartment for Rent</td>
</tr>
<tr>
<td>swp:FloridaAveStudio</td>
<td></td>
</tr>
</tbody>
</table>
Property paths can also be used to create a more concise SPARQL query using the | operator that can express one or more possibilities:

```
PREFIX swp: <http://www.semanticwebprimer.org/ontology/apartments.ttl#>.
SELECT ?apartment WHERE {

{?

  dbpedia-owl:location | dbpedia-owl:locationCity
  dbpedia:Amsterdam.}

OPTIONAL
{?

  rdfs:label ?label.}
}
```
Optional Patterns

<uni:lecturer rdf:about="949352">
  <uni:name>Grigoris Antoniou</uni:name>
</uni:lecturer>

<uni:professor rdf:about="94318">
  <uni:name>David Billington</uni:name>
  <uni:email>david@work.example.org</uni:email>
</uni:professor>

• For one lecturer it only lists the name
• For the other it also lists the email address
Optional Patterns

• All lecturers and their email addresses:

```
SELECT ?name ?email
WHERE
{ ?x rdf:type uni:Lecturer ;
  uni:name ?name ;
  uni:email ?email .
}
```

• Grigoris Antoniou is listed as a lecturer, but he has no e-mail address, so he is not selected

<table>
<thead>
<tr>
<th>name</th>
<th>email</th>
</tr>
</thead>
<tbody>
<tr>
<td>David Billington</td>
<td><a href="mailto:david@work.example.org">david@work.example.org</a></td>
</tr>
</tbody>
</table>
Optional Patterns

- As a solution we can adapt the query to use an optional pattern:

```sparql
SELECT ?name ?email
WHERE
{ ?x rdf:type uni:Lecturer ;
  uni:name ?name .
  OPTIONAL { x? uni:email ?email }
}
```

- The meaning is roughly “give us the names of lecturers, and if known also their e-mail address”

<table>
<thead>
<tr>
<th>?name</th>
<th>?email</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grigoris Antoniou</td>
<td></td>
</tr>
<tr>
<td>David Billington</td>
<td><a href="mailto:david@work.example.org">david@work.example.org</a></td>
</tr>
</tbody>
</table>
Organizing Result Sets

- It is often the case that we want the results of our queries to be returned in a particular way, either grouped, counted, or ordered:
  - We can eliminate duplicate results from the results set using the **DISTINCT** keyword by placing it after the **SELECT** keyword (this will ensure that only unique variable bindings are returned)
  - We can order a returned result set using the **ORDER BY** keyword
    - The keyword **DESC** denotes descending order.
    - Likewise, **ASC** denotes ascending order.
    - Ordering a string or url is done alphabetically.
Organizing Result Sets

- Find the apartments ordered by the number of bedrooms:

```sparql
SELECT ?apartment ?bedrooms
WHERE {
  ?apartment swp:hasNumberOfBedrooms ?bedrooms.
}
ORDER BY DESC(?bedrooms)
```

```
<table>
<thead>
<tr>
<th>?apartment</th>
<th>?bedrooms</th>
</tr>
</thead>
<tbody>
<tr>
<td>swp:BaronWayApartment</td>
<td>3</td>
</tr>
<tr>
<td>swp:FloridaAveStudio</td>
<td>1</td>
</tr>
</tbody>
</table>
```
Organizing Result Sets

- Collect results set together using *aggregate* functions
  - count the number of results (**COUNT**)
  - sum (**SUM**),
  - minimum, maximum, and average (**MIN**, **MAX**, **AVG**).

PREFIX swp: <http://www.semanticwebprimer.org/ontology/apartments.ttl#>.

SELECT (AVG(?bedrooms) AS ?avgNumRooms)
WHERE {
  ?apartment swp:hasNumberOfBedrooms ?bedrooms.
}

```
?avgNumRooms
2
```
Other Forms of SPARQL Queries

- ASK query simply checks to see whether a graph pattern exists in a data set instead of returning a result.

```sparql
PREFIX swp: <http://www.semanticwebprimer.org/ontology/apartments.ttl#>.
ASK ?apartment
WHERE {
  ?apartment swp:hasNumberOfBedrooms 3.
}
```

- ASK queries are used because they are faster to compute than retrieving an entire set of results.
Other Forms of SPARQL Queries

- The CONSTRUCT query is used to retrieve an RDF graph from a larger set of RDF, not a list of variable bindings

```sparql
PREFIX swp: <http://www.semanticwebprimer.org/ontology/apartments.ttl#>.
CONSTRUCT { ?apartment swp:hasNumberOfBedrooms ?bedrooms.
    ?apartment swp:isBigApartment true. }
WHERE{
    ?apartment swp:hasNumberOfBedrooms ?bedrooms.
}
FILTER (?bedrooms > 2)
```

- A graph is returned with new properties
Querying Schemas

- Consider an RDFS housing ontology

```ttl
@prefix swp: <http://www.semanticwebprimer.org/ontology/apartments.ttl#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.

swp:Unit rdf:type rdfs:Class.
swp:ResidentialUnit rdf:type rdfs:Class.

swp:ResidentialUnit rdfs:subClassOf swp:Unit.
swp:Apartment rdf:type rdfs:Class.
swp:Apartment rdfs:subClassOf swp:ResidentialUnit.
```
Determine the Residential Units in our dataset by querying both the instance data and schema simultaneously

```
PREFIX swp: <http://www.semanticwebprimer.org/ontology/apartments.ttl#>.
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
SELECT ?apartment
WHERE{
  ?apartment a ?unitType.
  ?unitType rdfs:subClassOf swp:ResidentialUnit.
}
```

- we used the same Turtle shorthand, `a`, to denote `rdf:type`
Adding Information with SPARQL Update

- SPARQL constructs for insertion, loading, and deleting of triples

```
PREFIX swp: <http://www.semanticwebprimer.org/ontology/apartments.ttl#>.
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
INSERT DATA
{
    swp:LuxuryApartment rdfs:subClassOf swp:Apartment.
}
- If you have a large file containing RDF available on the web, you can load it into a triple store using the following command:
LOAD <http://example.com/apartment.rdf>
```
Deleting Information with SPARQL Update

- Delete triples with DELETE DATA:
  
  PREFIX swp: <http://www.semanticwebprimer.org/ontology/apartments.ttl#>.
  PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
  DELETE DATA
  
  { 
    swp:LuxuryApartment rdfs:subClassOf swp:Apartment.
  } 

  - no variables are allowed and all triples must be fully specified

- Delete triples with DELETE WHERE:

  - remove all the triples containing information about apartments with more than two bedrooms

  PREFIX swp: <http://www.semanticwebprimer.org/ontology/apartments.ttl#>.
  PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
  DELETE WHERE{
    ?apartment swp:hasNumberOfBedrooms ?bedrooms.
    FILTER (?bedrooms > 2)
  }
Deleting Information with SPARQL Update

- Remove all the contents of a triple store the CLEAR construct:

  CLEAR ALL
References

- http://www.w3.org/TR/sparql11-query/
- http://www.w3.org/TR/sparql11-update/
- http://www.w3.org/TR/rdf-sparql-protocol/
- http://jena.sourceforge.net/ARQ/Tutorial/