Querying RDF with SPARQL

WebKR
18 February 2009
Eyal Oren, eyal@cs.vu.nl
Contents

- Reminder: RDF and RDF Schema
- Last week's exercise
- SPARQL: basic concepts and syntax
- SPARQL: querying schemas
Reminder: what is RDF?

- RDF: graph-based model for representing (meta)data
  - describe properties of resources
  - using URIs or literal values
    - literal: “Antoine Isaac”, “020^^xsd:integer”, ...

- You can write RDF in NTriples or Turtle
- Can write RDF in XML, advantage over 'normal' XML:
  - make interpretation explicit
  - agree on meaning of tags
Reminder: what is RDF Schema?

- RDF Schema standardises RDF vocabulary for describing classes and properties
  - Subclasses, subproperties, domain/range, ...
- These terms have formal semantics
  - “A sc B, B sc C → A sc C”
  - “X a B, B sc C → X a C”
- RDFS semantics specified by entailment rules
- RDF Schema: a simple ontology language
Reminder: RDF and RDF Schema

GeographicEntity

Country

EuropeanCountry

Country

City

Capital

Netherlands

Amsterdam

rdf:type

rdfs:subClassOf

rdfs:domain

rdfs:range

rdfs:subClassOf

rdfs:subClassOf

rdfs:domain

rdfs:range

rdfs:subClassOf

rdfs:subClassOf

hasCapital

hasCapital

AreaCode

“020″^^xsd:integer
Aside: language-tagged literals

- Literals with (XML) language tags

```
Paris a geo:City;
  geo:name “Paris”@fr, “Parijs”@nl .
```

```
<geo:City rdf:about="#Paris">
  <geo:name xml:lang="fr">Paris</geo:name>
  <geo:name xml:lang="nl">Parijs</geo:name>
</geo:city>
```
Last Week’s Assignment

- Create RDF(S) model with at least 3 classes and properties, and some instances.

- Use properties like `type`, `subClassOf`, `subPropertyOf`, `domain`, `range` correctly.

- Write your RDF(S) model as Turtle file, make sure it validates for syntactical correctness.
Exercise: Example Solution

Diagram:

- Animal
  - subClassOf: Cat
  - subClassOf: Dog
  - subClassOf: Donkey

- Cat
  - range: chases
  - domain: type
  - name: "Tom"

- Dog
  - range: chases
  - domain: type
  - name: "Lassie"

- Donkey
  - range: chases
  - domain: type
  - name: "Eddy"
Example solution in Turtle

@prefix : <http://example.org/animals#> .
:Dog a rdfs:Class ; rdfs:subClassOf :Animal .
:Cat a rdfs:Class ; rdfs:subClassOf :Animal .
:Donkey subClassOf :Animal .
:chases rdfs:domain :Dog ; rdfs:range :Cat .

:tom a :Cat; :name “Tom” .
:eddie a :Donkey .
:lassie a :Dog ; :name “Lassie” ; :chases ex:tom .
How should I pick my URIs?

- which URIs should I use? Who checks that these URIs “exist”? Should I “declare” all URIs?
- You can use whatever you want: RDF doesn't care. If *other people* should understand it: use standardised vocabularies (DublinCore, FOAF)

```
@prefix g: <http://google.com/rdf#> .
g:eyal a g:Person .
```

Is this allowed? Can I use their namespace? Shouldn't there be some RDF there? Will anyone understand this? Shouldn't I put RDF there?
Linked data principles (linkeddata.org)

- Use URIs as names for things
- Use HTTP URIs so people can lookup stuff
- Provide useful descriptions at your HTTP URIs
- Include links to other URIs

- Go to http://dbpedia.org/resource/Amsterdam
  - You'll get some RDF describing Amsterdam

- Go to http://xmlns.com/foaf/0.1/knows
  - You'll get RDF describing foaf:knows
How should I pick my URIs?

- how many namespaces should I use?
- You can use whatever you want: RDF doesn't care. If *other people* should understand it: separate it into coherent pieces. You don't need to separate properties vs classes.

@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix : <http://example.org/foaf#> .

:eyal a foaf:Person;
   foaf:knows :stefan .
How do I create instances?

ex:Dog a rdfs:Class ; rdfs:subClassOf ex:Animal .
ex:eye rdfs:domain ex:Animal ;
    rdfs:range xsd:string .
ex:fikkie a ex:Dog ; ex:eye “blue” .
Can you write rules in RDFS?

- A subClassOf B, B subClassOf C → A subClassOf C
  - Yes
- A a Dog, A age 14 → A a VeryOldDog
  - No

- RDFS is not a rule language
- RDFS contains some built-in rules, but not more
Namespaces again

- Is this correct or wrong?
  - Syntactically? Will it parse correctly?
  - Semantically? Is “hasTeacher” now defined as subProperty?

```turtle
hasAssistantTeacher a rdf:Property;
  rdfs:subPropertyOf hasTeacher .
```
Revealed modelling weakness

In the schema:

```
:hasColor a rdf:Property ;
  rdfs:domain :Car, :Motorcycle, :Airplane;
  rdfs:range :Color .
```

In the data:

```
:Ferrari a :Car; hasColor :Red .
```

What is the type of Ferrari?
Car and Motorcycle and Plane
Summary: modelling in RDF(S)

- RDF is easy: just invent terms as you want
- RDF is hard: stick to the syntax

- RDFS is easy: just subclasses and properties
- RDFS is hard: stick to the semantics

- And now: let's query all this data!
Contents

- Reminder: RDF and RDF Schema
- Last week's exercise
- SPARQL: basic concepts and syntax
- SPARQL: querying schemas
Do you remember SQL?

- Formulate a query on the relational model
  - students(name, age, address)

- Structured Query Language (SQL)
  ```sql
  SELECT name  
  FROM student  
  WHERE age > 20
  ```

<table>
<thead>
<tr>
<th>name</th>
<th>age</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>21</td>
<td>Amsterdam</td>
</tr>
</tbody>
</table>
SPARQL

- Standard RDF query language
  - based on existing ideas
  - standardised by W3C
  - widely supported
- Standard RDF query protocol
  - how to send a query over HTTP
  - how to respond over HTTP
- Can SPARQL also query OWL data?
SPARQL Query Syntax

SPARQL uses a select-from-where inspired syntax (like SQL):

- **select**: the entities (variables) you want to return
  
  SELECT ?city

- **from**: the data source (RDF dataset)
  
  FROM <http://example.org/geo.rdf>

- **where**: the (sub)graph you want to get the information from
  
  WHERE {?city geo:areacode “010” .}

- Including additional constraints on objects, using operators
  
  WHERE {?city geo:areacode ?c. FILTER (?c > 010)}

- **prologue**: namespace information
  
  PREFIX geo: <http://example.org/geo.rdf>
PREFIX geo: <http://example.org/geo/>

SELECT ?city
FROM <http://example.org/geoData.rdf>
     FILTER (?c > 010) 
}
SPARQL Graph Patterns

The core of SPARQL

- WHERE clause specifies graph pattern
  - pattern should be matched
  - pattern can match more than once

- Graph pattern:
  - an RDF graph
  - with some nodes/edges as variables

```
EuropeanCountry

type

hasCapital

? ?

“020”^^xsd:integer
```
Basis: triple patterns

- Triples with one/more *variables*
- Turtle syntax
  
  - ?X geo:hasCapital geo:Amsterdam
  - ?X geo:hasCapital ?Y
  - ?X geo:areacode "020"
  - ?X ?P ?Y

- All of them match this graph:
Basis: triple pattern

A very basic query

```sparql
PREFIX geo: <http://example.org/geo/>
SELECT ?X
FROM  <http://example.org/geoData.rdf>
WHERE { ?X geo:hasCapital ?Y .}
```
Conjunctions: several patterns

A pattern with several graphs, all must match

```
PREFIX geo: <http://example.org/geo/>
SELECT ?X
FROM <http://example.org/geoData.rdf>
WHERE { {?X geo:hasCapital ?Y } {?Y geo:areacode "020" } }
```

equivalent to

```
PREFIX geo: <http://example.org/geo/>
SELECT ?X
FROM <http://example.org/geoData.rdf>
WHERE { ?X geo:hasCapital ?Y . ?Y geo:areacode "020" . }
```
Conjunctions: several patterns

A pattern with several graphs, all must match

```sql
PREFIX geo: <http://example.org/geo/>
SELECT ?X
FROM <http://example.org/geoData.rdf>
WHERE {
   {?X geo:hasCapital ?Y }
   {?Y geo:areacode "020" } }
```
equivalent to

```sql
PREFIX geo: <http://example.org/geo/>
SELECT ?X
FROM <http://example.org/geoData.rdf>
WHERE {
   ?X geo:hasCapital [ geo:areacode "020" ].
}
```
Note: Turtle syntax again

- \(?X\) geo:name \(?Y\) ; geo:areacode \(?Z\)
- \(?X\) geo:name \(?Y\) . \(?X\) geo:areacode \(?Z\)
- ?country geo:capital [ geo:name "Amsterdam" ]
A pattern with several graphs, at least one should match

```sparql
PREFIX geo: <http://example.org/geo/>
SELECT ?city
WHERE {
  { ?city geo:name "Parijs"@nl }
UNION
  { ?city geo:name "Paris"@fr .}
}
```
**Optional Graphs**

- RDF is *semi*-structured
  - Even when the schema says some object can have a particular property, it may not always be present in the data
  - Example: persons can have names and email addresses, but Frank is a person without a known email address

```
person001
  +------------------+
  |                  |
  |      name       |
  |                  |
  |      email      |
  |                  |
  +------------------+

person002
  +------------------+
  |                  |
  |      name        |
  |                  |
  +------------------+
```

- Name: "Antoine"
- Email: "aisaac@few.vu.nl"
- Name: "Frank"
Optional Graphs (2)

- “Give me all people with first names, *and if known* their email address”
- An **OPTIONAL** graph expression is needed

```sparql
PREFIX : <http://example.org/my#>
SELECT ?person ?name ?email
WHERE {
  ?person :name ?name .
  OPTIONAL { ?person :email ?email } 
}
```
Testing values of nodes

Tests in FILTER clause have to be validated for matching subgraphs

- **RDF model-related operators**
  - `isLiteral(?aNode)`
  - `isURI(?aNode)`
  - `STR(?aResource)`

Interest of STR?

```
SELECT ?X ?N
  FILTER (STR(?P)="areacode") }
```

- For resources with names only partly known
- For literals with unknown language tags
Testing values of nodes

Tests in FILTER clause

- **Comparison:**
  - \(?X \leq ?Y, \ ?Z < 20, \ ?Z = ?Y, \ etc.\)

- **Arithmetic operators**
  - \(?X + ?Y, \ etc.\)

- **String matching** using regular expressions
  - REGEX (?X,"netherlands","i")
    - matches "The Netherlands"

PREFIX geo: <http://example.org/geo/>
SELECT ?X ?N
    FILTER REGEX(STR(?N),"dam") }
Filtering results

- Tests in **FILTER** clause
  - Boolean *combination* of these test expressions
    - && (and), || (or), ! (not)
    - $(?Y > 10 \land \land ?Y < 30)$
      - || !REGEX(?Z,"Rott")

PREFIX geo: <http://example.org/geo/>
SELECT ?X
FROM <http://example.org/geo.rdf>
WHERE { ?X geo:areacode ?Y ;
    geo:name ?Z .
FILTER ( (?Y > 10 \land \land ?Y < 30) ||
    !REGEX(STR(?X),"Rott")) }
Boolean comparisons and datatypes

- Reminder: RDF has basic datatypes for literals
  - XML Schema datatypes: `xsd:integer`, `xsd:float`, `xsd:string`, etc.

- Datatypes can be used in value comparison
  - `X < "21"^^xsd:integer`

- and be obtained from literals
  - `DATATYPE(?aLiteral)`
Solution modifiers

- ORDER BY

SELECT ?dog ?age
WHERE { ?dog a Dog ; ?dog :age ?age . }
ORDER BY DESC(?age)

- LIMIT

SELECT ?dog ?age
WHERE { ?dog a Dog ; ?dog :age ?age . }
ORDER BY ?dog
LIMIT 10
SELECT Query Results

- SPARQL SELECT queries return solutions that consist of *variable bindings*
  - For each variable in the query, it gives a value (or a list of values).
  - The result is a table, where each column represents a variable and each row a combination of variable bindings
Query result: example

- Query: “return all countries with the cities they contain, and their areacodes, if known”

```sparql
PREFIX geo: <http://example.org/geo/>
SELECT ?X ?Y ?Z
WHERE { ?X geo:containsCity ?Y.
    OPTIONAL {?Y geo:areacode ?Z} }
```

- Result (as table of bindings):

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>Netherlands</td>
<td>Amsterdam</td>
<td>“020”</td>
</tr>
<tr>
<td>Netherlands</td>
<td>DenHaag</td>
<td>“070”</td>
</tr>
</tbody>
</table>
SELECT Query results: format

- Query: return all capital cities
  ```
  PREFIX geo: <http://example.org/geo/>
  SELECT ?X ?Y
  WHERE { ?X geo:name ?Y .}
  ```

- Results as an XML document:
  ```xml
  <sparql xmlns="http://www.w3.org/2005/sparql-results#">
    <head>
      <variable name="X"/>
      <variable name="Y"/>
    </head>
    <results>
      <result>
        <binding name="X"><uri>http://example.org/Paris</uri></binding>
        <binding name="Y"><literal>Paris</literal></binding>
      </result>
      <result>
        <binding name="X"><uri>http://example.org/Paris</uri></binding>
        <binding name="Y"><literal xml:lang="nl">Parijs</literal></binding>
      </result>
      ...
    </results>
  </sparql>
  ```
Query Result forms

- SELECT queries return variable bindings

- Do we need something else?
  - Statements from RDF original graph
    - Data extraction
  - New statements derived from original data according to a specific need
    - Data conversion, views over data
SPARQL CONSTRUCT queries

- Construct-queries return RDF statements
  - The query result is either a **subgraph** of the original graph, or a **transformed** graph

Subgraph query:

```sparql
PREFIX geo: <http://example.org/geo/>
CONSTRUCT { ?X geo:hasCapital ?Y }
WHERE { ?X geo:hasCapital ?Y .
  ?Y geo:name "Amsterdam" }
```

Graph:

- Netherlands
  - hasCapital
  - Amsterdam
SPARQL CONSTRUCT queries

- Construct-queries return RDF statements
  - The query result is either a **subgraph** of the original graph, or a **transformed** graph

```
PREFIX geo: <http://example.org/geo/>
PREFIX my: <http://example.org/myNS/>
CONSTRUCT {?Y my:inCountry ?X}
WHERE { ?X geo:hasCapital ?Y}
```

**Transformation query:**

- Netherlands
- inCountry
- Amsterdam
SPARQL queries

- **SELECT: table (variable bindings)**
  
  ```
  select ?x where { ... }
  ```

- **CONSTRUCT: graph**
  
  ```
  construct { ... } where { ... }
  ```

- **ASK: yes/no**
  
  ```
  ask { ... }
  ```

- **DESCRIBE: graph**
  
  ```
  describe dbpedia:Amsterdam
  ```
Contents

- Reminder: RDF and RDF Schema
- What should an RDF query language do?
- SPARQL: basic concepts and syntax
- SPARQL: schema-related and advanced features
Schema Querying

- SPARQL has support for Schema querying
  - Class instances
  - Subclasses, Subproperties
  - etc.

- Remember: RDF Schemas are RDF graphs with special resources!
Schema Querying

GeographicEntity

subClassOf

Country

subClassOf

EuropeanCountry

hasCapital

Capital

subClassOf

City

subClassOf

hasCapital

Netherlands

Amsterdam

area code

"020"
Schema Querying

GeographicEntity

- subClassOf
  - Country
  - EuropeanCountry

- hasCapital
  - City

- type
  - Capital

- domain
  - City

- range
  - Capital

Ontology level

Data level

Netherlands

- hasCapital
  - Amsterdam

Type

Area code

“020”
Schema querying example

- Query: “return the range of the property `hasCapital`”

```
PREFIX geo: <http://example.org/geo/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?X
WHERE  {geo:hasCapital rdfs:range ?X .}
```

- Query: “return all subclasses of `GeographicEntity`”

```
PREFIX geo: <http://example.org/geo/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?X
WHERE  { ?X rdfs:subClassOf geo:GeographicEntity .}
```
Ontology/Data Querying

GeographicEntity

Country

EuropeanCountry

Netherlands

City

Capital

Amsterdam

ontology level

data level

hasCapital

subClassOf

domain

range

type

areacode

“020”
Query: “return all instances of the class Country”

```
PREFIX geo: <http://example.org/geo/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?X
WHERE { ?X rdf:type geo:Country .}
```
Ontology/Data Querying Example

- Query: “return all countries, and the assertions (properties and values) for each”

```rdf
PREFIX geo: <http://example.org/geo/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?X ?P ?Y
WHERE {
  FILTER (?P != rdf:type) }
```
Summary

- We need a specific query language for RDF and RDF Schema
  - XQuery won't do the job

- SPARQL is a language
  - Expressive
    - Path expressions, schema/data querying, etc.
  - Easy of use
  - Implemented
Assignment

See BlackBoard: formulate SPARQL queries over DBpedia data!
Links, further reading, etc.

- **SPARQL**
  - Site:
    http://www.w3.org/2001/sw/DataAccess/
  - Recommendation
    http://www.w3.org/TR/rdf-sparql-query/
  - Quick reference
    http://www.dajobe.org/2005/04-sparql/
  - Validator
    http://www.sparql.org/validator.html