Introduction to the Semantic Web

Lecture 5: The SPARQL query language

Basil Ell · Cord Wiljes

AG Semantic Computing

November 14, 2018

Schedule

10.10.2018	Lecture 1: Introduction - What is the Semantic Web?
17.10.2018	Lecture 2: The Resource Description Framework (RDF)
24.10.2018	Lecture 3: RDF Schema (RDFS)
07.11.2018	Lecture 4: RDF Interpretation
14.11.2018	**Lecture 5: SPARQL**
21.11.2018	Lecture 6: SPARQL query evaluation
28.11.2018	Lecture 7: OWL
05.12.2018	Lecture 8: Protégé
12.12.2018	Lecture 9: LOD
19.12.2018	Lecture 10: Vocabularies
09.01.2019	Lecture 11: Ontology Engineering, Alignment, Evaluation
16.01.2019	Lecture 12: Summary and Coaching
23.01.2019	Lecture 13: Presentations pt. 1
30.01.2019	Lecture 14: Presentations pt. 2

SPARQL: The query language for RDF

SPARQL

- SPARQL is the abbreviation for
  'SPARQL Protocol And RDF Query Language'
- is a language that allows to query RDF data
- is a W3C Recommendation since January 2008
- latest version: SPARQL 1.1 (May 2013)
- see `http://www.w3.org/TR/rdf-sparql-query/`
  for more details

Querying the Semantic Web

Let’s look at a concrete example of a SPARQL Query:

```
PREFIX ex: <http://example.org/>
SELECT ?title ?author
WHERE {
  ?book ex:publishedBy <http://springer.com/Verlag> .
  ?book ex:hasTitle ?title .
  ?book ex:hasAuthor ?author.
}
```

Result

title	author
”Semantic Web - Grundlagen”	`http://example.org/Hitzler`
”Semantic Web - Grundlagen”	`http://example.org/Krötzsch`
”Semantic Web - Grundlagen”	`http://example.org/Rudolph`
”Semantic Web - Grundlagen”	`http://example.org/Sure`

Optional Pattern (OPTIONAL)

```
PREFIX ex: <http://example.org/>
SELECT ?book ?title ?author
WHERE {
  ?book ex:publishedBy <http://springer.com/Verlag>
  OPTIONAL { ?book ex:hasTitle ?titel . }
  OPTIONAL { ?book ex:hasAuthor ?author . }
}
```

- OPTIONAL is left-associative
- Variables can be unbound (in the result) as a result of using OPTIONAL

Result

book	title	author
`http://example.org/DB_Intro`		`http://example.org/Somebody`
`http://example.org/SW_Foundations`	”Semantic Web Grundlagen”	`http://example.org/Hitzler`
`http://example.org/SW_Foundations`	”Semantic Web Grundlagen”	`http://example.org/Krötzsch`
`http://example.org/SW_Foundations`	”Semantic Web Grundlagen”	`http://example.org/Rudolph`
`http://example.org/SW_Foundations`	”Semantic Web Grundlagen”	`http://example.org/Sure`
`http://example.org/IR_Intro`	”Introduction to Inf. Retrieval”	

Alternative Pattern (UNION)

```
PREFIX ex: <http://example.org/>
SELECT ?book ?author
WHERE {
  ?book ex:publishedBy <http://springer.com/Verlag> .
  { ?book ex:hasAuthor ?author . } UNION
  { ?book ex:hasWriter ?author . }
}
```

UNION is a binary operator (left-associative)
UNION has precedence over Sequence

Filters

We can use so called filters to narrow down the result set:

```
PREFIX ex: <http://example.org/>
SELECT ?book
WHERE {
  ?book ex:publishedBy <http://springer.com/Verlag> .
  ?book ex:hasPrice ?price
  FILTER (?price < 35)
}
```

Comparison Operators

- Comparison operators: $<, =, >, <=, >=, ! =$
- Comparison of datatype literals w.r.t. their natural order
- Support for numerical datatypes (xsd:int, xsd:float etc.), xsd:dateTime, xsd:string (alphabetical order) and xsd:Boolean ($1 > 0$)
- For other types, only the = and != operators are defined
- Comparison of non-compatible types not supported (e.g. xsd:string with xsd:integer)

Boolean Operators (Filter)

- Logical AND (&&)
- Logical OR (||)
- Negation (!)

```
PREFIX ex: <http://example.org/>
SELECT ?book
WHERE {
  ?book ex:publishedBy <http://springer.com/Verlag> .
  ?book  ex:hasPrice ?price
  FILTER (?price > 10 && ?price < 100)
}
```

Special Predicates

BOUND(A)
isURI(A)
isBLANK(A)
isLiteral(A)
STR(A)
LANG(A)
DATATYPE(A)
REGEX(A,B)

Result Modifiers (ORDER BY)

- Results can be ordered with respect to some criterion
- Sort-result determined by comparison operators (as when using filters)
- URIs are sorted alphanumerically
- Other possibilities:
    - ORDER BY DESC(?price): decreasing
    - ORDER BY ASC(?price): increasing (default)
    - ORDER BY DESC(?price), ?title: additional sorting criteria in case of equality

Other types of SPARQL Queries

- CONSTRUCT
- ASK
- DESCRIBE

CONSTRUCT-Queries

```
PREFIX ex: <http://example.org>
CONSTRUCT {
  _:id1 ex:email ?email .
  _:id1 ex:telefon ?telefon .
  _:id1 ex:person ?person .
}
WHERE {
  ?person ex:email ?email .
  ?person ex:tel ?telefon
}
```

ASK-Queries

```
PREFIX ex: <http://example.org>
ASK {
  ?person ex:email ?email .
  ?person ex:tel ?telefon .
}
```

DESCRIBE-Queries

```
DESCRIBE <http://www.wiljes.de/cord>
```

Practical part - DBpedia

- DBpedia is a machine readable RDF version of Wikipedia infoboxes
- data from DBpedia was automatically extracted from Wikipedia - therefore contains some noise
- Contains resources e.g. `http://dbpedia.org/resource/Barack_Obama`
- Contains properties e.g.: `http://dbpedia.org/ontology/birthDate` `http://dbpedia.org/property/dateOfBirth`
- DBpedia has an own SPARL-endpoint `http://dbpedia.org/sparql`

Practical part - Example

Task: Given a natural question in natural language, go to DBpedia and try to create a SPARQL-query, which is equivalent to the question.

Question: When was Barack Obama born?

Practical part - Example

Task: Given a natural question in natural language, go to DBpedia and try to create a SPARQL-query, which is equivalent to the question.

Question: When was Barack Obama born?

Solution:

```
PREFIX res: <http://dbpedia.org/resource/>
PREFIX dbo: <http://dbpedia.org/ontology/>
SELECT ?date WHERE {
  res:Barack_Obama dbo:birthDate ?date .
}
```

Practical part

Task: For the following list of questions in natural language, go to DBpedia and try to create a SPARQL-query, which is equivalent to those questions.

These questions are part of the training dataset of the Question Answering over Linked Data challenge (QALD5).

When was Michelle Obama born?

```
PREFIX res: <http://dbpedia.org/resource/>
PREFIX dbo: <http://dbpedia.org/ontology/>
SELECT ?date WHERE {
  res:Michelle_Obama dbo:birthDate ?date .
}
```

In which country does the Ganges start?

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX res: <http://dbpedia.org/resource/>
SELECT DISTINCT ?uri
WHERE {
  res:Ganges dbo:sourceCountry ?uri .
}
```

Is proinsulin a protein?

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX res: <http://dbpedia.org/resource/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
ASK WHERE {
  res:Proinsulin rdf:type dbo:Protein .
}
```

How tall is Michael Jordan?

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX res: <http://dbpedia.org/resource/>
SELECT DISTINCT ?num
WHERE {
  res:Michael_Jordan dbo:height ?num .
}
```

What is the highest mountain in Australia?

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX res: <http://dbpedia.org/resource/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT DISTINCT ?uri
WHERE {
  ?uri rdf:type dbo:Mountain .
  ?uri dbo:locatedInArea res:Australia .
  ?uri dbo:elevation ?elevation .
}
ORDER BY DESC(?elevation)
OFFSET 0 LIMIT 1
```

Which German cities have more than 250.000 inhabitants?

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX res: <http://dbpedia.org/resource/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT DISTINCT ?uri
WHERE {
  { ?uri rdf:type dbo:City . }
   UNION
  { ?uri rdf:type dbo:Town . }
  ?uri dbo:country res:Germany .
  ?uri dbo:populationTotal ?population .
  FILTER ( ?population > 250000 )
}
```

Give me all cosmonauts.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX res: <http://dbpedia.org/resource/>
SELECT DISTINCT ?uri
WHERE {
  ?uri rdf:type dbo:Astronaut .
  { ?uri dbo:nationality res:Russia . }
  UNION
  { ?uri dbo:nationality res:Soviet_Union . }
}
```