Querying Semantic Web – SPARQL

Miroslav Blaško, Petr Křemen

miroslav.blaško@fel.cvut.cz, petr.kremen@fel.cvut.cz

October 10, 2019
Outline

1. SPARQL
   - SPARQL Basics
   - SPARQL Update (Graph Update Operations)
1. SPARQL
   - SPARQL Basics
   - SPARQL Update (Graph Update Operations)
A simple SPARQL Query

```
SELECT ?person {
  ?person a <http://xmlns.com/foaf/0.1/Person> .
} LIMIT 10
```

To be queried over RDF data inside a **SPARQL endpoint**, e.g. http://dbpedia.org/sparql
SPARQL idea

**SPARQL client**

```
SELECT ?person {
  ?person a /xmlns.com/foaf/0.1/Person>
} LIMIT 10
```

**SPARQL endpoint 1**

http://dbpedia.org/sparql

---

**SPARQL endpoint 2**

http://etree.linkedmusic.org/sparql

---

Each SPARQL client sends a query to one of the SPARQL endpoints, which then processes the query and returns results to the client.
SPARQL Factsheet

- **SPARQL 1.1** – 12 W3C Recommendations on 21 March 2013, covering
  - a query language (SPARQL 1.1 Query Language) [Harris:13:SQL]
SPARQL Factsheet

- SPARQL 1.1 – 12 W3C Recommendations on 21 March 2013, covering
  - a query language (SPARQL 1.1 Query Language) [Harris:13:SQL]
  - an update language (SPARQL 1.1. Update language)
SPARQL Factsheet

- SPARQL 1.1 – 12 W3C Recommendations on 21 March 2013, covering
  - a query language (SPARQL 1.1 Query Language) [Harris:13:SQL]
  - an update language (SPARQL 1.1. Update language)
  - SPARQL services (protocol over HTTP, graph management HTTP protocol),
**SPARQL Factsheet**

- **SPARQL 1.1** – 12 W3C Recommendations on 21 March 2013, covering
  - a query language (SPARQL 1.1 Query Language) [Harris:13:SQL]
  - an update language (SPARQL 1.1. Update language)
  - SPARQL services (protocol over HTTP, graph management HTTP protocol),
  - an extension for executing distributed queries over more SPARQL endpoints [Aranda:13:SFQ]
SPARQL Factsheet

- SPARQL 1.1 – 12 W3C Recommendations on 21 March 2013, covering
  - a query language (SPARQL 1.1 Query Language) [Harris:13:SQL]
  - an update language (SPARQL 1.1. Update language)
  - SPARQL services (protocol over HTTP, graph management HTTP protocol),
  - an extension for executing distributed queries over more SPARQL endpoints [Aranda:13:SFQ]
  - JSON, CSV, TSV, XML query result formats [Seaborne:13:SQR]
SPARQL Factsheet

- **SPARQL 1.1** – 12 W3C Recommendations on 21 March 2013, covering
  - a query language (SPARQL 1.1 Query Language) [Harris:13:SQL]
  - an update language (SPARQL 1.1. Update language)
  - SPARQL services (protocol over HTTP, graph management HTTP protocol),
  - an extension for executing distributed queries over more SPARQL endpoints [Aranda:13:SFQ]
  - JSON, CSV, TSV, XML query result formats [Seaborn:13:SQR]
  - definition of entailment regimes for RDF extensions (e.g. OWL, more in lecture 10) [Ogbuji:13:SER].
SPARQL for RDF is like SQL for RDBMS

'Get projects having male administrators starting on the letter N'

```
SELECT e.surname AS es,
    p.name AS pn
FROM employee e, project p
WHERE e.gender = 'male'
    AND p.administratorId = e.id
    AND e.surname LIKE 'N\%';
```

```
PREFIX : <http://example.org/>
SELECT ?sn, (?projname AS ?pn)
WHERE {
    ?e a :Employee .
    ?e :gender 'male'.
    ?p a :Project .
    FILTER (strstarts(?sn,'N'))
}
```

However, SPARQL is less powerful comparing to SQL in terms of built-in functions
Is SPARQL the only one?

Some previous attempts to query SPARQL include:
- reactive-rule languages – e.g. Algea
- path-based languages – e.g. Versa
- relational-based – TRIPLE, Xcerpt, SeRQL

At present

SPARQL is **The standard** for querying RDF. In addition, graph languages have been gaining popularity for querying RDF (e.g. GraphQL [https://graphql.org/](https://graphql.org/)).
SPARQL Basics
Query Types

**SELECT** – returns a binding table (similarly to SQL)
Query Types

**SELECT** – returns a binding table (similarly to SQL)

**ASK** – returns a true/false indicating existence of the given pattern in the RDF graph
Query Types

**SELECT** – returns a binding table (similarly to SQL)

**ASK** – returns a true/false indicating existence of the given pattern in the RDF graph

**CONSTRUCT** – returns an RDF graph constructed from the binding table
Query Types

**SELECT** – returns a binding table (similarly to SQL)

**ASK** – returns a true/false indicating existence of the given pattern in the RDF graph

**CONSTRUCT** – returns an RDF graph constructed from the binding table

**DESCRIBE** – returns an RDF graph describing the given resource (semantics not fixed)
Query Evaluation

Pattern matching

\{ ?e a :Employee .
?e :gender 'male' .\}

SELECT ?e
\{
?e a :Employee .
?e :gender 'male' .
\}

ASK
\{
?e a :Employee .
?e :gender 'male' .
\}

CONSTRUCT \{
?e a :MaleEmployee .
\} 
\{
?e a :Employee .
?e :gender 'male' .
\}

has result

Binding Table

?e

<http://example.org/johnsmith>

<http://example.org/garysmith>

ASK true/false result
true

RDF Graph

@prefix : <http://example.org/> :johnsmith a :Employee .
:johnsmith :gender 'male' .
:marysmith a :Employee .
:marysmith :gender 'female' .
:susannesmith a :Employee .
:garysmith a :Employee .
:garysmith :gender 'male' .

has result

has result

RDF Graph

@prefix : <http://example.org/> :johnsmith a :MaleEmployee .
:johnsmith :gender 'male' .
:marysmith a :MaleEmployee .
:marysmith :gender 'male' .
:susannesmith a :MaleEmployee .
:susannesmith :gender 'male' .
:garysmith a :MaleEmployee .
:garysmith :gender 'male' .
Basic Definitions (1)

**RDF Term** $\in T = T_I \cup T_B \cup T_L$, being a union of set of all IRIs, blank nodes and literals respectively.

**example**

\[::_a < http://example.org/data/John > "John"@en\]

**solution** is a mapping $\mu : V \rightarrow T$ assigning an RDF term to each variable from the query,

\[\mu = \{(?person \rightarrow < http://example.org/data/John >),
          (?personName \rightarrow "John"@en)\}\]

**result set** is a list $R = (\mu_1, \ldots, \mu_n)$ of solutions,
**Basic Definitions (2)**

**triple pattern (TP)** is a member of $(T \cup V) \times (T_l \cup V) \times (T \cup V)$,

<table>
<thead>
<tr>
<th>example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>(?person, a, foaf:Person)</code></td>
</tr>
</tbody>
</table>

or in the turtle syntax

| ?person a foaf:Person |

**basic graph pattern (BGP)** is a set $BGP = \{ TP_1, \ldots, TP_n \}$ of triple patterns.

<table>
<thead>
<tr>
<th>example</th>
</tr>
</thead>
</table>

**graph store** is a mutable container providing an RDF dataset at each time,
Basic Graph Patterns

Repository content:

```sparql
@prefix : <http://example.org/>  
@prefix r: <http://dbpedia.org/resource/>  
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>

prefix inventor {  
  r:Thomas_Edison :invented :bulb .  
  r:J_Cimrman :invented :bulb .  
  :wheel rdfs:label "Wheel"@en .  
  :x :invented :wheel .  
  :y :invented :SteamEngine .  
  :z :invented :Gunpowder .  
  :Gunpowder rdfs:label "Strelny prach"@cs .  
}
```

Query with a BGP

```sparql
PREFIX : <http://example.org/>  
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?s ?l
WHERE {
  ?s :invented ?i.
  ?i rdfs:label ?l.
}
```

Table: Result set

<table>
<thead>
<tr>
<th>s</th>
<th>l</th>
</tr>
</thead>
<tbody>
<tr>
<td>r:Thomas_Edison</td>
<td>“Bulb”@en</td>
</tr>
<tr>
<td>r:J_Cimrman</td>
<td>“Bulb”@en</td>
</tr>
<tr>
<td>r:Thomas_Edison</td>
<td>“Zarovka”@cs</td>
</tr>
<tr>
<td>r:J_Cimrman</td>
<td>“Zarovka”@cs</td>
</tr>
<tr>
<td>_:a</td>
<td>“Wheel”@en</td>
</tr>
<tr>
<td>_:b</td>
<td>“Strelny prach”@cs</td>
</tr>
</tbody>
</table>
Filtering results

**Description**

<table>
<thead>
<tr>
<th>syntax</th>
<th>BGP1 <strong>FILTER</strong>(boolean condition) BGP1</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td><strong>FILTER</strong> clause filters BGP results (anywhere in a BGP)</td>
</tr>
</tbody>
</table>

**Query with a BGP**

```
PREFIX : <http://example.org/>
PREFIX rdfs: ← <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?s ?l
WHERE {
  ?s :invented ?i.
  ?i rdfs:label ?l
  FILTER(regex(?l,"^.*ul\.*")
        && contains(str(?s),"Cimr"))
}
```

- **string functions** – e.g. strlen, contains, substr, concat, regex, replace
- **RDF term functions** – e.g. isIRI, IRI, isBlank, BNODE, isLiteral, str, lang, datatype

See SPARQL 1.1 spec.
https://www.w3.org/TR/2013/REC-sparql11-query-20130321/
Graph Patterns – Overview

Graph patterns cover all basic algebraic operations:

- conjunction (sequence of graph patterns),
- disjunction (\texttt{UNION} pattern),
- negation (\texttt{FILTER NOT EXISTS, MINUS})
- conditional conjunction (\texttt{OPTIONAL})
Optional data

Description

syntax GP1 OPTIONAL { GP2 }

description results of GP1 are optionally augmented with results of GP2, if any. Optionals are left-associative.

Two optionals

```sparql
PREFIX : <http://example.org/>
PREFIX rdfs:
← <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?s ?i ?l
WHERE {
  ?s :invented ?i.
  OPTIONAL {
    ?i rdfs:label ?l FILTER (lang(?l)="en").
  }
  OPTIONAL {
    ?i rdfs:label ?l FILTER (lang(?l)="cs")
  }
}
```

Table: Result set

<table>
<thead>
<tr>
<th>s</th>
<th>l</th>
</tr>
</thead>
<tbody>
<tr>
<td>r:Thomas_Edison</td>
<td>“Bulb”@en</td>
</tr>
<tr>
<td>r:J_Cimrman</td>
<td>“Bulb”@en</td>
</tr>
<tr>
<td>:a</td>
<td>“Wheel”@en</td>
</tr>
<tr>
<td>:b</td>
<td>“Strelny prach”@cs</td>
</tr>
</tbody>
</table>
Other examples

**FILTERing with regular expressions**

```sparql
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?title
  ?x dc:author ?author
FILTER regex(?title, ".SPARQL") }
```

**Order of OPTIONALs might be important**

```sparql
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX books: <http://books.example.org>
SELECT ?writing ?name
WHERE {
```
Negation

negation as failure – i.e. what cannot be inferred is considered false.

two constructs – **MINUS** vs. **FILTER NOT EXISTS**

**MINUS**

```sparql
PREFIX : <http://example.org/>
PREFIX rdfs:
<http://www.w3.org/2000/01/rdf-schema#>

SELECT ?s1 ?i
{   ?s1 :invented ?i.
    MINUS {
        ?s2 :invented ?i.
        FILTER(?s1 != ?s2).
    } }
```

Variable ?s1 is not bound in the **MINUS** pattern. Returns all inventors.

**FILTER NOT EXISTS**

```sparql
PREFIX : <http://example.org/>
PREFIX rdfs:
<http://www.w3.org/2000/01/rdf-schema#>

SELECT ?s1 ?i
{   ?s1 :invented ?i.
    FILTER NOT EXISTS {
        ?s2 :invented ?i.
        FILTER(?s1 != ?s2).
    } }
```

Returns all inventions that were invented just by one inventor.
Property Paths

### Description

Property paths allow to express simple regular expressions on properties, as follows:

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Matches ((e(i)) means path element, (p(i)) means (iri) or (^{^}iri))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(iri)</td>
<td>an IRI (path of length 1)</td>
</tr>
<tr>
<td>(^e)</td>
<td>an inverse path (o (\rightarrow) s)</td>
</tr>
<tr>
<td>(e_1 \ / \ e_2)</td>
<td>a sequence path of (e_1) followed by (e_2)</td>
</tr>
<tr>
<td>(e_1 \</td>
<td>\ e_2)</td>
</tr>
<tr>
<td>(e^*)</td>
<td>a sequence path of zero or more matches of (e)</td>
</tr>
<tr>
<td>(e+)</td>
<td>a sequence path of one or more matches of (e)</td>
</tr>
<tr>
<td>(e?)</td>
<td>a sequence path of zero or one more matches of (e)</td>
</tr>
<tr>
<td>(!\left((p_1]</td>
<td>\ldots</td>
</tr>
<tr>
<td>((e))</td>
<td>group path (brackets for precedence)</td>
</tr>
</tbody>
</table>
Property Paths – Examples

Get the name of a resource

```sparql
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT *
{
}
```

Get elements of an RDF collection

```sparql
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT *
{
    ?s (rdf:rest*)/rdf:first ?listItem.
}
```
Aggregations

Description

Similarly to SQL, SPARQL allows using aggregation functions for numeric/string data:

\textbf{COUNT}(\var)\, or \textbf{COUNT}(\text{DISTINCT } \var) – counts number of (distinct) occurrences of \var in the resultset,

\textbf{MIN}(\var), \textbf{MAX}(\var), \textbf{SUM}(\var), \textbf{AVG}(\var) – analogous to their SQL counterparts,

\textbf{GROUP\_CONCAT}(\var; \text{separator }= <\text{SEP}>) \text{ AS } \text{?group}) – concatenates all elements in the group with the given separator character,

\textbf{SAMPLE} – takes an arbitrary representative from the group.

Usage of (\text{?expr as } \var) alias is obligatory.

Similarly to SQL, SPARQL allows computing aggregates over particular data groups and filter in them using \textbf{GROUP\ BY}/\textbf{HAVING} construct.
Aggregation – Examples

Compute the number of inventors of each invention.

```sparql
PREFIX : <http://example.org/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT (COUNT(?s) as ?count) ?i (GROUP_CONCAT(?s;separator=",") as ?inventors)
FROM :inventors
WHERE {
  ?s :invented ?i.
}
GROUP BY ?i
HAVING (COUNT(?s) > 1)
```
**Variable assignment**

**Description**

Variables can be assigned results of function (or aggregation function). The syntax is $(expr \ AS \ ?v)$, where $expr$ is an expression and $?v$ is the newly create variable not appearing before.

Compute the number of inventions of each inventor.

```sparql
PREFIX : <http://example.org/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT (COUNT(?s) AS ?count) ?invention
FROM :inventors
WHERE {
  ?s :invented ?i .
  ?i rdfs:label ?l
  BIND (concat("Invention: ",?l) AS ?invention)
}
GROUP BY ?i ?invention
```
### Distributed Queries

#### Syntax and semantics

**syntax**  
\[ \ldots \text{SERVICE (SILENT)} \text{sparqlServiceURI} \{ \text{GP} \} \]

**semantics**  
This clause poses a sparql query described by graph pattern GP to a remote SPARQL endpoint `sparqlServiceURI`.

#### DBPedia service query

```sql
PREFIX : <http://example.org/>
PREFIX p: <http://dbpedia.org/property/>
PREFIX r: <http://dbpedia.org/resource/>
SELECT ?s ?p ?o ?i
WHERE {
  GRAPH :inventors { ?s :invented ?i. }
  OPTIONAL { SERVICE SILENT
    <http://dbpedia.org/sparql> { 
      ?s ?p ?o
      FILTER ( strstarts(str(?p),
        concat(str(p:),"death")) ) } } }
```

#### Local repo content

```sql
@prefix : <http://example.org/>
@prefix p: <http://dbpedia.org/property/>
@prefix r: <http://dbpedia.org/resource/>

:inventors {
  r:Thomas_Edison :invented :bulb.
  r:J_Cimrman :invented :bulb.
}
Selected Other Features

- **VALUES** – predefined variable binding specified in the tabular form
Selected Other Features

- **VALUES** – predefined variable binding specified in the tabular form
- **ORDER BY, LIMIT, OFFSET** – used analogously to SQL
Selected Other Features

- **VALUES** – predefined variable binding specified in the tabular form
- **ORDER BY, LIMIT, OFFSET** – used analogously to SQL
- **FROM, FROM NAMED** – used to specify active default/named graphs for the query
Selected Other Features

- **VALUES** – predefined variable binding specified in the tabular form
- **ORDER BY, LIMIT, OFFSET** – used analogously to SQL
- **FROM, FROM NAMED** – used to specify active default/named graphs for the query
- **SELECT DISTINCT** – removes duplicates from the results
**SPARQL Entailment Regimes**

- simply – SPARQL spec. [Harris:13:SQL] defines evaluation of BGPs w.r.t. *simple entailment*
- [Ogbuji:13:SER] defines a several other entailment regimes for SPARQL BGPs:
  - RDF entailment, RDFS entailment, D-entailment, as defined in RDF spec.
  - OWL 2 entailments, RIF entailment, that are the much more expressive, see lecture 10.

... conditions for defining custom entailment regimes

---

**All SPARQL entailment regimes must ensure**

- compliance with the corresponding entailment (e.g. RDF, RDFS)
- finiteness of results
  - only *canonical* b-nodes can be returned (ensured by skolemization of both the query and the queried graph),
  - only finite part of respective vocabularies can be returned as query results (e.g. RDF vocabulary without `rdf:n` properties not occurring)
SPARQL Evaluation Semantics

PREFIX : <http://ex.org/e1>
SELECT ?x
WHERE { ?x :p :d }
SPARQL Evaluation Semantics

\[
\text{PREFIX} : \ <\text{http://ex.org/e1}> \\
\text{SELECT} \ \?x \\
\text{WHERE} \ \{ \ ?x :p :d \} \\
\]

Simple-entailment No result.

RDF-entailment No result.
SPARQL Evaluation Semantics

```
PREFIX : <http://ex.org/e1>
SELECT ?x
WHERE { ?x :p :d }
```

Simple-entailment  No result.

RDF-entailment  No result.

RDFS-entailment One result: \( ?x = :a \).
SPARQL Evaluation Semantics

```
PREFIX : <http://ex.org/e1>
SELECT ?x
WHERE { ?x :p :d }
```

**Simple-entailment**  No result.

**RDF-entailment**  No result.

**RDFS-entailment**  One result:  ?x= :a.

CSV for SELECT; loses information about datatypes/languages of RDF terms

TSV for SELECT; is lossless

XML, JSON for SELECT, ASK; is lossless, supports additional information (e.g. columns identification through link attribute),

```json
{
  "head": {
    "vars": [ "person", "name" ]
  },
  "results": {
    "bindings": [
      {
        "person": {
          "type": "uri",
          "value": "http://ex.com/p1"
        },
        "name": {
          "type": "literal",
          "value": "Smith"
        }
      },
      {
        "person": {
          "type": "uri",
          "value": "http://ex.com/p2"
        }
      }
    ]
  }
}
```
Related Technologies

**SPIN** (SPARQL inference notation) – SPARQL rules encoded in RDF (http://spinrdf.org/)
Related Technologies


Related Technologies


**SNORQL** – Web front-end for exploring SPARQL endpoints ([https://github.com/kurtjx/SNORQL](https://github.com/kurtjx/SNORQL))
Related Technologies

**SPIN** (SPARQL inference notation) – SPARQL rules encoded in RDF ([http://spindrdf.org/](http://spindrdf.org/))


**SNORQL** – Web front-end for exploring SPARQL endpoints ([https://github.com/kurtjx/SNORQL](https://github.com/kurtjx/SNORQL))

**SeRQL** – Sesame query language (alternative to SPARQL)
Related Technologies


**SNORQL** – Web front-end for exploring SPARQL endpoints ([https://github.com/kurtjx/SNORQL](https://github.com/kurtjx/SNORQL))

**SeRQL** – Sesame query language (alternative to SPARQL)

**SQWRL** (Semantic Query-Enhanced Web Rule Language) – query language based on SWRL (see next lecture), [http://protege.cim3.net/cgi-bin/wiki.pl?SQWRL](http://protege.cim3.net/cgi-bin/wiki.pl?SQWRL)
SPARQL Update (Graph Update Operations)
Inserting

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
INSERT { <http://example/person> dc:title "John" } 
WHERE { }
```
Deleting

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
DELETE WHERE
  ?person a foaf:Person .
}
Replacing

```sparql
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dbo: <http://dbpedia.org/ontology/>
DELETE { ?person a foaf:Person . }
INSERT { ?person a dbo:Person . }
WHERE { ?person a foaf:Person . }
```
Other operations

- **LOAD** – loading a graph into a graph store
- **CLEAR** – clearing a graph inside a graph store
- **CREATE** – create a new graph in a graph store
- **DROP** – deletes a graph in a graph store
- **COPY** – inserts all triples from one graph to another, clearing the dest.
- **MOVE** – moves all triples from one graph to another
- **ADD** – inserts all triples from one graph to another, keeping the dest.

See [https://www.w3.org/TR/sparql11-update/](https://www.w3.org/TR/sparql11-update/) for details