SPARQL

Petr Křemen

petr.kremen@fel.cvut.cz

October 7, 2020
Outline

1 SPARQL
   - SPARQL Query Language Basics
   - SPARQL Update (Graph Update Operations)
SPARQL

- SPARQL Query Language 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/Possession
} LIMIT 10

SPARQL endpoint 1
http://dbpedia.org/sparql

SPARQL endpoint 2
http://etree.linkedmusic.org/sparql

evaluates query in
triple store
evaluates query in
triple store
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 \[Seaborne:13:SQR]\]
  - 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'

```sparql
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\%';
```

```sparql
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, or subqueries
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.
SPARQL Query Language 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)
Select Evaluation

RDF Graph

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

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

Binding Table

<table>
<thead>
<tr>
<th>?e</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://example.org/johnsmith">http://example.org/johnsmith</a></td>
</tr>
<tr>
<td><a href="http://example.org/garysmith">http://example.org/garysmith</a></td>
</tr>
</tbody>
</table>
Ask Evaluation

RDF Graph
@prefix : <http://example.org/>
:johnsmith a :Employee ;
    :gender 'male'.
:marysmith a :Employee ;
    :gender 'female'
:susannsmith a :Employee .
:garysmith a :Employee ;
    :gender 'male'.

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

result is non-empty?
true/false

Matching

Binding Table

<table>
<thead>
<tr>
<th>?e</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://example.org/johnsmith">http://example.org/johnsmith</a></td>
</tr>
<tr>
<td><a href="http://example.org/garysmith">http://example.org/garysmith</a></td>
</tr>
</tbody>
</table>
Construct Evaluation

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

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

Matching

Binding Table
<table>
<thead>
<tr>
<th>?e</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

creates RDF graph according to a pattern

RDF Graph
@prefix : <http://example.org/>
:johnsmith a :MaleEmployee .
:garysmith a :MaleEmployee .
Query Solutions

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.

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

result list is a list \( R = (\mu_1, \ldots, \mu_n) \) of solutions,

example

**Graph:**

\[ :\text{John} :\text{hasName} "\text{John}@en \]

**Query:**

```
SELECT ?person ?personName {?person :hasName ?name}
```

**Solution:**

\[ \mu = \{ (?person\rightarrow :\text{John}), (?name\rightarrow "\text{John}@en") \} \]
Graph Patterns

triple pattern (TP) is a member of \((T \cup V) \times (T_I \cup V) \times (T \cup V)\),

example

\((?\text{person}, a, \text{foaf:Person})\)

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.

example

?person a foaf:Person .
Basic Graph Patterns

Repository content

@prefix : <http://example.org/>
@prefix r: <http://dbpedia.org/resource/>
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
:Bulb rdfs:label "Bulb"@en , "Zarovka"@cs .
:Wheel rdfs:label "Wheel"@en .
:Gunpowder rdfs:label "Strelny prach"@cs .

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.
}

Results

<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>
</tbody>
</table>
Basic Graph Patterns

Repository content

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

r:Thomas_Edison :invented :Bulb .
r:J_Cimrman :invented :Bulb .
:Bulb rdfs:label "Bulb"@en , "Zarovka"@cs .
:Wheel rdfs:label "Wheel"@en .
:_:x :invented :Wheel .
:_:y :invented :SteamEngine .
:_:z :invented :Gunpowder .
:Gunpowder rdfs:label "Strelny prach"@cs .
```

Results

```
<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>
```

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.}
```
Filtering results

**Description**

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

**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  
  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** as a sequence of graph patterns,
- **disjunction** as `UNION`,
- **negation** as `FILTER NOT EXISTS` or `MINUS`
- **conditional conjunction** as `OPTIONAL`
Optional data

Description

**syntax**  \[
GP1 \text{ OPTIONAL } \{ \text{ GP2 } \}
\]

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

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

<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></td>
</tr>
<tr>
<td>_:c</td>
<td>“Strelny prach”@cs</td>
</tr>
</tbody>
</table>

Result set
FILTERing with regular expressions

```sparql
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?title
WHERE {
  ?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
{
  OPTIONAL {
  } .
  OPTIONAL {
  }
}
```
Negation

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

MINUS

Variable $s_1$ is not bound in the MINUS pattern. Returns all inventors.

FILTER NOT EXISTS

Returns all inventions that were invented just by one inventor.

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

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

### Description

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

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Matches</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>iri</code></td>
<td>an IRI (path of length 1)</td>
</tr>
<tr>
<td><code>^e</code></td>
<td>an inverse path (o → s)</td>
</tr>
<tr>
<td><code>e₁ / e₂</code></td>
<td>a sequence path of <code>e₁</code> followed by <code>e₂</code></td>
</tr>
<tr>
<td>`e₁</td>
<td>e₂`</td>
</tr>
<tr>
<td><code>e*</code></td>
<td>a sequence path of zero or more matches of <code>e</code></td>
</tr>
<tr>
<td><code>e+</code></td>
<td>a sequence path of one or more matches of <code>e</code></td>
</tr>
<tr>
<td><code>e?</code></td>
<td>a sequence path of zero or one more matches of <code>e</code></td>
</tr>
<tr>
<td>`!(p₁</td>
<td>...</td>
</tr>
<tr>
<td><code>(e)</code></td>
<td>group path (brackets for precedence)</td>
</tr>
</tbody>
</table>
Get the name of a resource

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

SELECT *

{" ?s rdfs:label | dc:title ?name. }
```
Get elements of an RDF collection

```
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:

- `COUNT(?var)`, or `COUNT(DISTINCT ?var)` – counts number of (distinct) occurrences of `?var` in the resultset,

- `MIN(?v), MAX(?v), SUM(?v), AVG(?v)` – similar to their SQL counterparts,

- `GROUP_CONCAT(?var; separator = <SEP>) AS ?group)` – concatenates all elements in the group with the given separator character,

- `SAMPLE` – takes an arbitrary representative from the group.

Usage of `(?expr AS ?var)` alias is obligatory.

Similarly to SQL, SPARQL allows computing aggregates over particular data groups and filter in them using `GROUP BY/HAVING` construct.
Compute the number of inventors of each invention.

```
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)
WHERE

{ ?s :invented ?i. }

GROUP BY ?i
HAVING (COUNT(?s) > 1)
```
Compute the number of inventions of each inventor.

Description

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

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

Syntax and semantics

**syntax** ... **SERVICE** (**SILENT**) **sparqlServiceURI** { **GP** }

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

DBPedia service query

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

Local repo content

```
@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.
  }
```
Other Features

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

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

- SPARQL [Harris:13:SQL] defines evaluation of BGPs w.r.t. *simple entailment*

- [Ogbuji:13:SER] defines 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 more expressive (refer to OWL lecture).
  - ... 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 in the graph).
SPARQL Evaluation Semantics

Simple-entailment No result.

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

**PREFIX**: <http://ex.org/e1>

**SELECT** ?x

**WHERE** { ?x :p :d }

**Simple-entailment** No result.

**RDF-entailment** No result.
SPARQL Evaluation Semantics

Simple-entailment  No result.
RDF-entailment   No result.
RDFS-entailment  One result:  \(?x = :a\).
**SPARQL Evaluation Semantics**

```sparql
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 SELECT/ASK results**

**CSV** for **SELECT**; loses information about datatypes/langs 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


**SHACL** – Shapes Constraint Language ([https://shacl.org/playground/](https://shacl.org/playground/))
Related Technologies


**iSPARQL** – SPARQL visual query builder  

**SHACL** – Shapes Constraint Language  
([https://shacl.org/playground/](https://shacl.org/playground/))

**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

```sparql
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