Querying Semantic Web – SPARQL

Petr Křemen

petr.kremen@fel.cvut.cz

March 7, 2014
Outline

1 Querying Semantic Web – SPARQL
   • SPARQL Query Language
Querying Semantic Web – SPARQL

SPARQL Query Language
History of RDF Query Languages

relational-based – SPARQL, RQL, TRIPLE, Xcerpt, SeRQL
History of RDF Query Languages

relational-based – SPARQL, RQL, TRIPLE, Xcerpt, SeRQL
reactive-rule language Algea (see http://www.w3.org/2001/Annotea)
History of RDF Query Languages

relational-based – SPARQL, RQL, TRIPLE, Xcerpt, SeRQL
reactive-rule language Algea (see http://www.w3.org/2001/Annotea)
  - *actions* (ask, assert, fwrule), *answers* (bindings for vars, proof RDF triples)
History of RDF Query Languages

relational-based – SPARQL, RQL, TRIPLE, Xcerpt, SeRQL
reactive-rule language Algea (see http://www.w3.org/2001/Annotea)
  • actions (ask, assert, fwrule), answers (bindings for vars, proof RDF triples)
path-based language Versa (see http://4suite.org)
History of RDF Query Languages

relational-based – SPARQL, RQL, TRIPLE, Xcerpt, SeRQL

reactive-rule language Algea (see http://www.w3.org/2001/Annotea)
  - actions (ask, assert, fwrule), answers (bindings for vars, proof RDF triples)

path-based language Versa (see http://4suite.org)
  - “XPath for RDF”
History of RDF Query Languages

relational-based – SPARQL, RQL, TRIPLE, Xcerpt, SeRQL

reactive-rule language Algea (see http://www.w3.org/2001/Annotea)
  - actions (ask, assert, fwrule), answers (bindings for vars, proof RDF triples)

path-based language Versa (see http://4suite.org)
  - “XPath for RDF”
  - forward/backward traversal, filtering, but no support for restructuring/constructing queries.
History of RDF Query Languages

relational-based  – SPARQL, RQL, TRIPLE, Xcerpt, SeRQL
reactive-rule language Algea (see http://www.w3.org/2001/Annotea)
  - actions (ask, assert, fwrule), answers (bindings for vars, proof RDF triples)
path-based language Versa (see http://4suite.org)
  - “XPath for RDF”
  - forward/backward traversal, filtering, but no support for restructuring/constructing queries.

... , there are plenty of them, but today SPARQL wins.
SPARQL vs. SQL

First, let’s shortly compare a query in SQL and SPARQL.

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

$$\text{SELECT } e.\text{surname AS es, p.name AS pn}$$
$$\text{FROM employee e, project p}$$
$$\text{WHERE e.gender = 'male'}$$
$$\text{AND p.administratorId = e.id}$$
$$\text{AND e.surname LIKE 'N\%'};$$
SPARQL Factsheet

- SPARQL 1.1 was standardized as a set of 12 W3C Recommendations on 21 March 2013, covering
  - a query language (SPARQL 1.1 Query Language) [?]
SPARQL Factsheet

- SPARQL 1.1 was standardized as a set of 12 W3C Recommendations on 21 March 2013, covering
  - a query language (SPARQL 1.1 Query Language) [?]
  - a “data definition language” (SPARQL 1.1. Update language)
SPARQL Factsheet

- SPARQL 1.1 was standardized as a set of 12 W3C Recommendations on 21 March 2013, covering
  - a query language (SPARQL 1.1 Query Language) [?]
  - a “data definition language” (SPARQL 1.1. Update language)
  - definition of SPARQL services (protocol over HTTP, graph management HTTP protocol), semantic description,
SPARQL Factsheet

- SPARQL 1.1 was standardized as a set of 12 W3C Recommendations on 21 March 2013, covering
  - a query language (SPARQL 1.1 Query Language) [?]
  - a “data definition language” (SPARQL 1.1. Update language)
  - definition of SPARQL services (protocol over HTTP, graph management HTTP protocol), semantic description,
  - an extension for executing distributed queries over more SPARQL endpoints [?]
SPARQL Factsheet

- SPARQL 1.1 was standardized as a set of 12 W3C Recommendations on 21 March 2013, covering
  - a query language (SPARQL 1.1 Query Language) [?]
  - a “data definition language” (SPARQL 1.1. Update language)
  - definition of SPARQL services (protocol over HTTP, graph management HTTP protocol), semantic description,
  - an extension for executing distributed queries over more SPARQL endpoints [?]
  - JSON, CSV, TSV, XML query result formats [?]
SPARQL Factsheet

- SPARQL 1.1 was standardized as a set of 12 W3C Recommendations on 21 March 2013, covering
  - a query language (SPARQL 1.1 Query Language)
  - a “data definition language” (SPARQL 1.1. Update language)
  - definition of SPARQL services (protocol over HTTP, graph management HTTP protocol), semantic description,
  - an extension for executing distributed queries over more SPARQL endpoints
  - JSON, CSV, TSV, XML query result formats
  - definition of entailment regimes for RDF extensions (e.g. OWL, see the respective lecture).
SPARQL Query Language
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

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'.

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

Pattern matching
{ ?e a :Employee.
 ?e :gender 'male'.
}

is used for
 executes
 produces

Binding Table

?e

<http://example.org/johnsmith>

<http://example.org/garysmith>

has result

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

has result

true

ASK true/false result

has result

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

has result
Basic Definitions

**RDF Term** is an element of the set of RDF terms \( T = T_I \cup T_B \cup T_L \), being a union of set of all IRIs, blank nodes and literals respectively.

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

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

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

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

**basic graph pattern (BGP)** is a set \( BGP = \{TP_1, \ldots, TP_n\} \) of triple patterns.
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})
Listing 1: Repository content

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

Listing 2: 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**

**syntax** BGP1 `FILTER(boolean condition)` BGP1

**description** `FILTER` clause filters BGP results; it can be anywhere in a BGP (does not break it)

---

**Listing 3 : 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.
Optional data

Description

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

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

---

*Listing 4 : 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></td>
</tr>
<tr>
<td>_:c</td>
<td>“Strelny prach” @cs</td>
</tr>
</tbody>
</table>

Petr Křemen (petr.kremen@fel.cvut.cz)  March 7, 2014  13 / 27
Other examples

Listing 5: FILTERing with regular expressions

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

Listing 6: 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**

**Listing 7 : 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.

**Listing 8 : 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>Description</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>
Property Paths – Examples

**Listing 9**: 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 *
{
}
```

**Listing 10**: 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:

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

- **MIN (?v)**, **MAX (?v)**, **SUM (?v)**, **AVG (?v)** – analogous 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.
Listing 11: 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) ?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 \((\text{expr} \ \text{AS} \ \?v)\), where \(\text{expr}\) is an expression and \(?v\) is the newly create variable not appearing before.

*Listing 12*: 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** ... **SERVICE (SILENT) sparqlServiceURI { GP }**

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

---

**Listing 13 : DBPedia service query**

```sparql
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")) ) } }
}
```

**Listing 14 : Local repo content**

```sparql
@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. \[?]\ defines evaluation of BGPs w.r.t. simple entailment

- \[?]\ 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 next 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.
SPARQL Evaluation Semantics

Simple-entailment  No result.
RDF-entailment  No result.

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

Simple-entailment  No result.

RDF-entailment  No result.

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

**Simple-entailment** No result.

**RDF-entailment** No result.

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

**OWL-entailment** Two results: \(?x = a\) and \(?x = b\).
SPARQL SELECT/ASK results

**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" }
                },
                 {
                    "person": {
                        "type": "uri",
                        "value": "http://ex.com/p2" }
                }
            ]
        }
    }
}
```
Related Technologies

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://spinrdf.org/)

**iSPARQL**  – SPARQL visual query builder (http://oat.openlinksw.com/isparql/)

**SNORQL**  – Web front-end for exploring SPARQL endpoints (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)
References