

SPARQL

Petr Křemen

`petr.kremen@cvut.cz`

Winter 2024



Outline

1 SPARQL

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



1

SPARQL

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

SPARQL



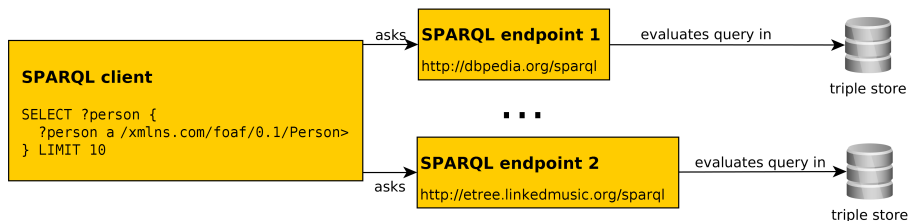
A simple SPARQL Query

```
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 mission



Factsheet

- SPARQL 1.1 – W3C Recommendations on 21 March 2013, covering



Factsheet

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



Factsheet

- SPARQL 1.1 – 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)



Factsheet

- SPARQL 1.1 – 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),



Factsheet

- SPARQL 1.1 – 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**]



Factsheet

- SPARQL 1.1 – 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**]



Factsheet

- SPARQL 1.1 – 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**].



Factsheet

- SPARQL 1.1 – 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 1.2 – W3C Working Draft (September 2024) – mainly support for RDF 1.2 (triple term patterns)



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 :surname ?sn .
  ?e :gender 'male'.
  ?p a :Project .
  ?p :name ?pn .
  ?p :administrator ? e.
  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

1

SPARQL

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



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)

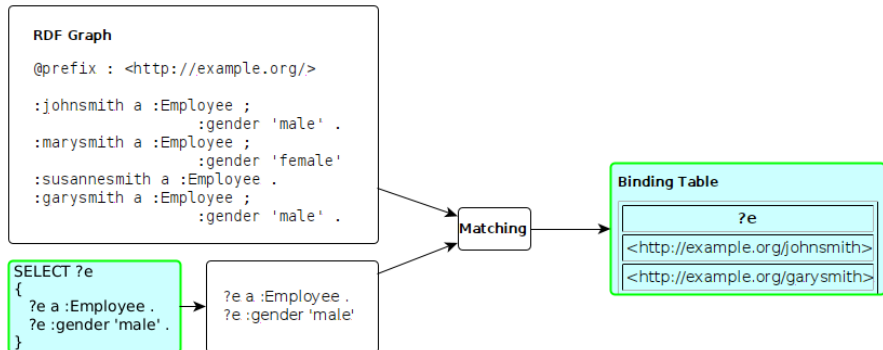
ASK – returns a true/false indicating existence of the given pattern in the RDF graph

CONSTRUCT – returns an RDF graph constructed from the binding table

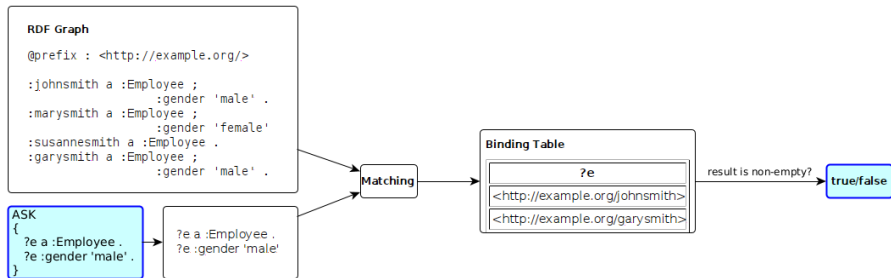
DESCRIBE – returns an RDF graph describing the given resource (semantics not fixed)



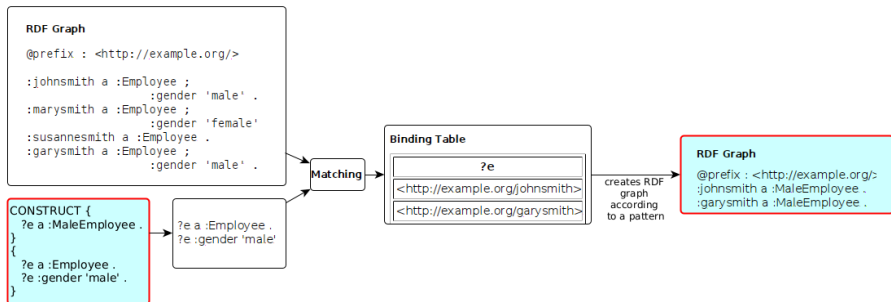
Select Evaluation



Ask Evaluation



Construct Evaluation



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 $R = (\mu_1, \dots, \mu_n)$ of solutions,

example

Graph:

```
:John :hasName "John"@en
```

Query:

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

Solution:

```
 $\mu = \{(?person \rightarrow :John), (?name \rightarrow "John"@en)\}$ 
```

Graph Patterns

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

example

```
(?person, a, foaf:Person)
```

or in the turtle syntax

```
?person a foaf:Person .
```

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

example

```
?person a foaf:Person .  
?person rdfs:label ?label .
```

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 .
:Gunpowder rdfs:label "Strelny prach"@cs .

```

Results

s	l
r:Thomas_Edison	"Bulb"@en
r:J_Cimrman	"Bulb"@en
r:Thomas_Edison	"Zarovka"@cs
r:J_Cimrman	"Zarovka"@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.
}

```



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

s	l
r:Thomas_Edison	"Bulb"@en
r:J_Cimrman	"Bulb"@en
r:Thomas_Edison	"Zarovka"@cs
r:J_Cimrman	"Zarovka"@cs
_:a	"Wheel"@en
_:b	"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.}

```



Filtering results

Description

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

description **FILTER** clause filters BGP results (anywhere in a BGP)

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

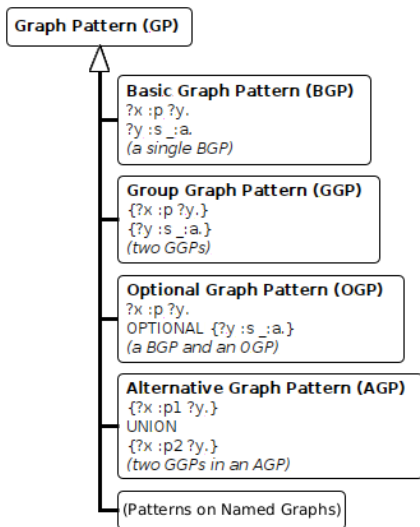
Graph patterns cover all basic algebraic operations:

conjunction 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 **OPTIONAL** { GP2 }

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

```

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")
  }
}

```

Result set

s	l
r:Thomas_Edison	"Bulb"@en
r:J_Cimrman	"Bulb"@en
_:a	"Wheel"@en
_:b	
_:c	"Strelny prach"@cs



FILTERing with regular expressions

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



Order of OPTIONALS might be important

PREFIX **rdf**:

↪ `<http://www.w3.org/1999/02/22-rdf-syntax-ns#>`

PREFIX **books**: `<http://books.example.org>`

SELECT ?writing ?name

WHERE

```
{  
  ?writing rdf:type books:Essay .  
  OPTIONAL {  
    ?writing books:translator ?p .  
    ?p dc:name ?name .  
  } .  
  OPTIONAL {  
    ?writing books:author ?p .  
    ?p dc:name ?name .  
  }  
}
```



Negation

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

MINUS

```
...
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

```
...
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.



MINUS vs. FILTER NOT EXISTS – another example

```
SELECT *  
{  
  ?s ?p ?o  
  MINUS  
    { ?x ?y ?z }  
}
```

Returns ALL results.

```
SELECT *  
{  
  ?s ?p ?o  
  FILTER NOT EXISTS  
    { ?x ?y ?z }  
}
```

Returns NO results.



Property Paths

Description

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

syntax	matches ($e_{(i)}$ means path element, $p_{(i)}$ means <i>iri</i> or \hat{iri})
<i>iri</i>	an IRI (path of length 1)
\hat{e}	an inverse path ($o \rightarrow s$)
e_1 / e_2	a sequence path of e_1 followed by e_2
$e_1 e_2$	an alternative path of e_1 or e_2
e^*	a sequence path of zero or more matches of e
e^+	a sequence path of one or more matches of e
$e?$	a sequence path of zero or one more matches of e
$!(p_1 \dots p_n)$	any IRI not matching any of p_i
(e)	group path (brackets for precedence)



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 (rdfs:rest*)/rdfs: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



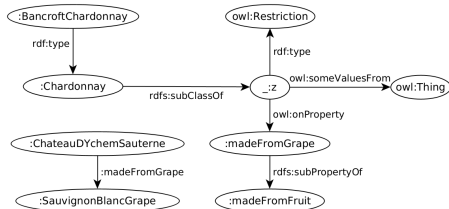
SPARQL Entailment Regimes

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

SPARQL Evaluation Semantics



```

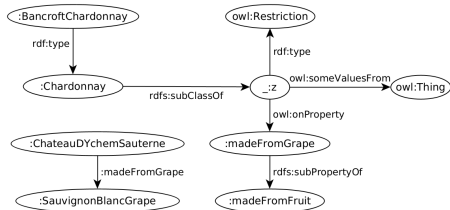
PREFIX : <http://ex.org/e1>
SELECT ?x
WHERE { ?x :madeFromFruit _:d }

```

Simple-entailment No result.



SPARQL Evaluation Semantics



```

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

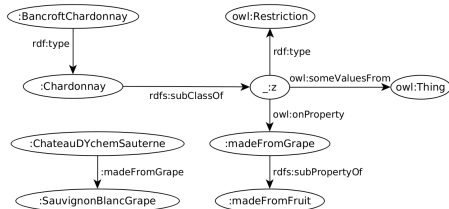
```

Simple-entailment No result.

RDF-entailment No result.



SPARQL Evaluation Semantics



```

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

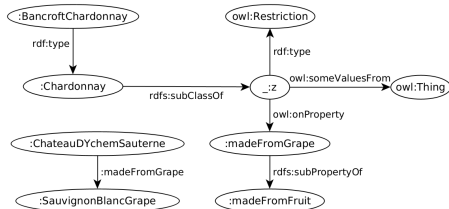
Simple-entailment No result.

RDF-entailment No result.

RDFS-entailment One result: `?x=:ChateauDYchemSauterne`.



SPARQL Evaluation Semantics



```

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

```

Simple-entailment No result.

RDF-entailment No result.

RDFS-entailment One result: ?x=:ChateauDYchemSauterne.

OWL-entailment Two results: ?x=:ChateauDYchemSauterne and
?x=:BancroftChardonnay.



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),

```
{
  "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

SPIN (SPARQL inference notation) – SPARQL rules encoded in RDF (<http://spinrdf.org/>)

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



Related Technologies

SPIN (SPARQL inference notation) – SPARQL rules encoded in RDF (<http://spinrdf.org/>)

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

SHACL – Shapes Constraint Language
(<https://shacl.org/playground/>)



Related Technologies

SPIN (SPARQL inference notation) – SPARQL rules encoded in RDF (<http://spinrdf.org/>)

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

SHACL – Shapes Constraint Language
(<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>



SPARQL Update (Graph Update Operations)

1

SPARQL

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



Inserting

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

or simply

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



Deleting

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

or simply

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



Replacing

```
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/> for details

