Web sémantique

SPARQL

olivier.corby@inria.fr

Plan

1. RDF : Resource Description Framework
2. RDFS : RDF Schema
3. SPARQL : RDF Query Language
4. OWL : Web Ontology Language
5. Examen

Agenda

1. SPARQL Query Language
2. SPARQL Update Language

Questions to RDF Graph

- Which resources does John know ?
- How old is James ?
- What is the name of John ?
- What is the subject of foaf:age ?
- Which property relates John and James ?
- Which property has 21 as value ?
- What are the properties of John ?
- …

SPARQL

- W3C Recommendation
- Simple Protocol And RDF Query Language
- SPARQL 1.0 Query (2008)
- SPARQL 1.1 Query (2013)
  - https://www.w3.org/TR/sparql11-query/
- SPARQL 1.1 Update (2013)
  - https://www.w3.org/TR/sparql11-update/

SPARQL Syntax

Triple Pattern

- Turtle triple syntax
- Variables

?x a foaf:Person

?x foaf:name "John"

<http://example.org/John> foaf:name ?name

<http://example.org/John> ?p ?v

Select Query

```
select *
where {
        ?x a foaf:Person .
}
```


Select Query

```
select *
where {
        ?x a foaf:Person ;
         foaf:name "John" .
}
```


Select Query

```
select *
where {
        ?x a foaf:Person ;
         foaf:name "John", ?name .
}
```


Select Query

```
select *
where {
        ?x a foaf:Person ;
         foaf:name "John", ?name .
        ?y foaf:knows ?x
}
```


Prefix, Namespace

```
prefix foaf: <http://xmlns.com/foaf/0.1/>
select *
where {
  ?x a foaf:Person ;
    foaf:name ?name .
}
```

Prefix, Namespace

```
select *
where {
  ?x a <http://xmlns.com/foaf/0.1/Person> ;
    <http://xmlns.com/foaf/0.1/name> ?name .
}
```


Blank Node

- *Anonymous* variable
- Value of Blank Node is <u>not returned</u> in result

```
select *
where {
        [ ] a foaf:Person ;
         foaf:name ?name
}
```


RDF List

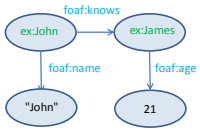
- Retrieve a list with the exact number of elements

```
select * where {
        ?x rdf:value (?y ?z)
}
```

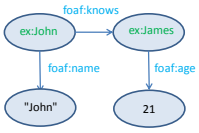

SPARQL Query Processing

- Query is a Graph Pattern (Graph with variables)
- Search occurrences of Graph Pattern in RDF Graph
- Zero, one or several results

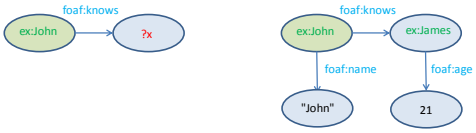
Graph Pattern Matching



Graph Pattern Matching

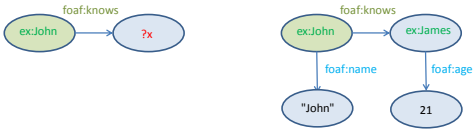


Graph Pattern Matching



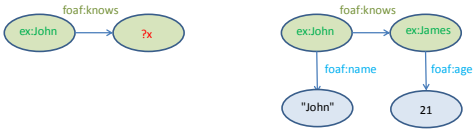
19

Graph Pattern Matching



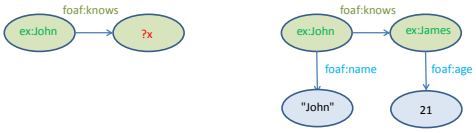
20

Graph Pattern Matching



21

Graph Pattern Matching

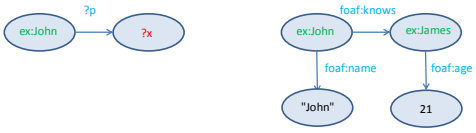


ex:John foaf:knows ?x

(1) ?x = ex:James

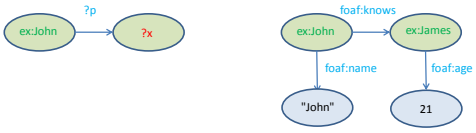
22

Graph Pattern Matching (2)



23

Graph Pattern Matching (2)

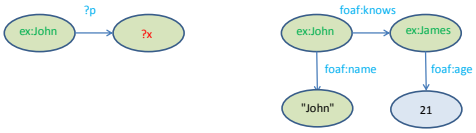


ex:John ?p ?x

(1) ?p = foaf:knows ; ?x = ex:James

24

Graph Pattern Matching (2)



ex:John ?p ?x

(1) ?p = foaf:knows ; ?x = ex:James
(2) ?p = foaf:name ; ?x = "John"

SPARQL Query Processing

- Result: (multi) set of variable bindings

1. ?x = ex:John ; ?y = ex:Jack
2. ?x = ex:Jim ; ?y = ex:Jesse
3. ?x = ex:John ; ?y = ex:Jack

Distinct Results

- Remove similar results with same values for same variables

```
select ?x ?z
where {
        ?x foaf:knows ?y .
        ?y foaf:knows ?z
}
```

1. ?x = ex:John ; ?z = ex:Jack
2. …
3. ?x = ex:John ; ?z = ex:Jack

Distinct Results

- Remove similar results with same values for same variables

```
select distinct ?x ?z
where {
        ?x foaf:knows ?y .
        ?y foaf:knows ?z
}
```

1. ?x = ex:John ; ?z = ex:Jack
2. …
3. ~~?x = ex:John ; ?z = ex:Jack~~

Example

- Resources with name "Iannis" and possibly other name(s)

Example

- Resources with name "Iannis" and possibly other name(s)

?x foaf:name "Iannis" , ?name .

Example

- Triple with same subject and object

Example

- Triple with same subject and object

?x ?p ?x

Example

- Triple with property as subject

Example

- Triple with property as subject

?p ?p ?x

Filter

Filter Expression

- Reduce result multiset
- Keep results that match a condition
- Match: expression evaluates to *true*

?age >= 10
?name != "John "
isURI(?x) || isBlank(?x)
! us:boring(?course)

Filter

- Variable:
  1. bound by triple pattern

```
select * where {
  ?x foaf:age ?age

}
```


Filter

- Variable:
  1. bound by triple pattern
  2. tested by filter

```
select * where {
  ?x foaf:age ?age
  filter (?age >= 18)
}
```


Filter Language

• URI, Literal, Variable	ex:John, 3.14, ?x
• < <= >= >	?age >= 18, ?n < "John"
• = !=	?x != ?y, ?y = "James"
• ( )	
• + - * /	(?n * (?n + 1))/2
• && || !	! (?x < 0 && ?y < 0)
• Function	datatype(?x)

Example

- Resources with name "Iannis" and with another name

Example

- Resources with name "Iannis" and with another name

```
?x foaf:name "Iannis" ,  ?name .
filter (?name != "Iannis")
```


Example

- Resources with different values for same property

Example

- Resources with different values for same property

?x ?p ?y , ?z .
filter (?y != ?z)

Example

- Resources with same value for different properties

Example

- Resources with same value for different properties

?x ?p ?v ; ?q ?v .
filter (?p != ?q)

Conditional Statement

- if then else

```
filter if (lang(?name) = "fr",
           ?age >= 18,
           ?age >= 21)
```


Function

- isBlank(?x)
- isURI(?x)
- isLiteral(?x)
- bound(?x)

Function

- datatype(?x)
- lang(?l)
- langMatches(lang(?l), "fr")
- str(<http://example.org>)
- uri("http://example.org")
- xsd:integer("123")
- xsd:string(12)
- strdt(str, datatype)
- strlang(str, lang)

Function

- now()
- year(?date)
- month(?date)
- day(?date)
- hours(?date)
- minutes(?date)
- seconds(?date)
- …

Function

- contains(str_1, str_2)
- strstarts(str_1, str_2)
- strends(str_1, str_2)
- concat(str_1, str_2)
- substr(str, n)
- strlen(str)
- regex(str, ".*cnrs")
- …

Function

- abs(?x)
- ceil(?x)
- floor(?x)
- round(?x)
- rand()
- …

Example

- Teenager : age between 13 and 19

Example

- Teenager : age between 13 and 19

```
?x foaf:age ?age
filter (?age >= 13  &&  ?age <= 19)
```


Example

- Resource whose name length is less than 20

Example

- Resource whose name length is less than 20

```
?x foaf:name ?name
filter (strlen(?name) < 20)
```


Example

- URI from inria

Example

- URI from inria

```
?x ?p ?y
filter regex(str(?x), "inria")
```


Query Form

Query Form

1. Select
2. Ask
3. Construct
4. Describe

Select

- Return variable bindings

```
select * where {
        ?x foaf:knows ?y
}
```

Ask

- Return true/false

ask { ?x rdf:type ex:Yeti }

Construct

- Return new RDF graph

1. Instantiate construct pattern with every results
2. Merge result triples into one new RDF graph

construct { ?x rdfs:seeAlso ?z }
where { ?x foaf:knows ?y . ?y foaf:knows ?z }

Describe

- Return description of resource(s) as RDF graph

describe <http://example.org/John>

describe ?x
where { ?x foaf:name "John" }

Statement

Statement

1. Union
2. Optional
3. Minus
4. (Not) Exists
5. Named Graph
6. Property Path
7. Nested Query
8. Bind
9. Values
10. Service

Union

- Union of results of alternative graph patterns
- One must match
- Both may match

{ ?x a ex:Good }
union
{ ?x a ex:Evil }

Optional

- Part of a query that is not mandatory
- Optional part may fail

?x a foaf:Person
optional { ?x foaf:name ?name }

1. ?x = us:John
2. ?x = us:Jack ; ?name = "Jack"

Minus

- Negation
- Remove result of minus pattern from result of graph pattern

?x a foaf:Person
minus { ?x foaf:age ?age . filter (?age < 18) }

Exists

- Check the existence of a pattern: return true or false

?x a foaf:Person
filter exists { ?y foaf:knows ?x }

Not Exists

- Check the absence of a pattern: return true or false

?x a foaf:Person
filter not exists { ?y foaf:knows ?x }

RDF Dataset

- One default graph (mandatory)
- Several named graphs (possibly)
- { G, (uri_1, G_1), .. (uri_n, G_n) }

- Basic Graph Pattern queries the default graph

From

- Redefine the default graph, for the query, either as:
  - one named graph
  - merge of several named graphs

select *
from ex:g1 { G, (ex:g1, G_1), .. (ex:g2, G_n) }
from ex:g2
where {
?x a foaf:Person $G_1 \cup G_2$
}

Named Graph Pattern

- Focus on one named graph

```
graph ex:cnrs  {
        ?x foaf:knows ?y
}
```


From named

- Query specific named graphs

```
select *
from named ex:cnrs
from named ex:uns
where {

        graph ?g  { ?x foaf:knows ?y }
}
```


From and From named

```
select *
from ex:inria
from named ex:cnrs
from named ex:uns
where {
        ?x a foaf:Person
        graph ?g  { ?x foaf:knows ?y }
}
```


Property Path

Find resources related by several triples:

ex:John $\xrightarrow{\text{foaf:knows}}$ ex:Jack $\xrightarrow{\text{foaf:knows}}$ ex:Jim $\xrightarrow{\text{foaf:knows}}$ ex:James

ex:John foaf:knows ?y

(1) ?y = ex:Jack

Property Path

Find resources related by several triples:

ex:John $\xrightarrow{\text{foaf:knows}}$ ex:Jack $\xrightarrow{\text{foaf:knows}}$ ex:Jim $\xrightarrow{\text{foaf:knows}}$ ex:James

ex:John foaf:knows/foaf:knows ?y

(1) ?y = ex:Jim

Property Path

Find resources related by several triples:

ex:John $\xrightarrow{\text{foaf:knows}}$ ex:Jack $\xrightarrow{\text{foaf:knows}}$ ex:Jim $\xrightarrow{\text{foaf:knows}}$ ex:James

ex:John foaf:knows/foaf:knows/foaf:knows ?y

(1) ?y = ex:James

Property Path

Find resources related by several triples:

ex:John —foaf:knows→ ex:Jack —foaf:knows→ ex:Jim —foaf:knows→ ex:James

ex:John foaf:knows+ ?y

(1) ?y = ex:Jack
(2) ?y = ex:Jim
(3) ?y = ex:James

Property Path Language

?x exp ?y

exp ::=

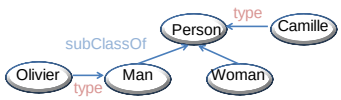
uri :	property
!uri :	negation
exp/exp :	sequence
exp|exp :	alternative
exp? :	optional
exp* :	zero or several
exp+ :	one or several
^exp :	reverse

Path Example

- Emulate class subsumption

?x rdf:type foaf:Person

?x rdf:type/rdfs:subClassOf* foaf:Person



Path Example

- Enumerate RDF list elements

?list rdf:rest*/rdf:first ?e

Nested Query

- Subquery within a query

```
select * where {
    {select ?x  (?w * ?l  as  ?area)
     where { ?x ex:width ?w ; ex:length ?l }}
    ?z ex:area ?area
}
```


Bind

- Bind result of expression to variable

```
?x geo:width ?w ; geo:length ?l
bind  (?w * ?l  as  ?area)
```

Values

- Focus variable(s) on predefined constant values

```
select * where {
        ?x rdfs:label ?name
}
values  ?name  {  "Blue"  "Red"  "Yellow"  }
```


Service

- Query remote SPARQL endpoint

```
select * where {
 service  <http://fr.dbpedia.org/sparql> {
        ?x  rdfs:label  "Nice"@fr ;
           rdf:type  ?t
 }

}
```


Result & Modifier

Result & Modifier

1. Select Expression
2. Order By
3. Group By
4. Aggregate
5. Having
6. Limit
7. Offset
8. Result Format

Select Expression

- Complete select clause with expressions

```
select  ?s  (?w * ?l  as  ?area)
where {
        ?s ex:width ?w ; ex:length ?l
}
```


Order By

- Sort results by numbers, strings, dates, …

```
select * where {
  ?x foaf:name ?name ;
     foaf:age ?age
}
order by desc(?age) ?name
```

Aggregate

- Combine several results into an aggregate value.

```
select (avg(?p) as ?avg)  where {
        ?x ex:price ?p
}
```

1. count
2. sum
3. avg
4. min
5. max
6. group_concat
7. sample

Aggregate & Group By

- Group results with similar values, aggregate in each group

```
select ?x  (avg(?s) as ?score)
where {
        ?x ex:score ?s
}
group by ?x
```


Aggregate, Group By, Having

- Group results with similar values, aggregate in each group, filter aggregate values

```
select ?x  (avg(?s) as ?score)
where {
        ?x ex:score ?s
}
group by ?x
having (?score >= 10)
```


Limit, Offset

- Limit number of results
- Slice results with an offset

```
select * where {
}
limit 10
offset 10
```


SPARQL Update

SPARQL Update

Manage (modify) content of RDF Dataset

- Load
- Insert/Delete Data
- Insert/Delete Where
- Copy, Move, Add

Load RDF document

```
load <http://example.org/data.ttl>

load <http://example.org/data.ttl>
into graph ex:g1
```


Insert Data

```
insert data {
    ex:John a foaf:Person ; foaf:age 18 .
    ex:Jill a foaf:Person ; foaf:age 81 .
}
```


Delete Data

```
delete data {
    ex:John a foaf:Person ; foaf:age 18 .
    ex:Jill a foaf:Person ; foaf:age 81 .
}
```


Delete Where

```
delete {
    ?x foaf:age ?age
}
where {
    ?x a foaf:Person ; foaf:age ?age
    filter (?age < 0)
}
```


Insert Where

```
insert {
    ?x foaf:mail ?mail
}
where {
    ?x ex:firstName ?f ; ex:lastName ?l
    bind (concat(?f, ".", ?l, "@acme.com")
    as ?mail)
}
```


Update: Delete Insert Where

```
delete { ?x foaf:age ?age }
insert  { ?x foaf:age ?int }
where {
    ?x foaf:age ?age
    filter (datatype(?age) = xsd:string)
    bind (xsd:integer(?age) as ?int)
}
```