SPARQL

SPARQL Protocol and RDF Query Language

Publishing information on the Semantic Web

- Data Access → SPARQL
- Information Organisation → RDFS, OWL
- Information format → RDF
- Identification → URIs
- Serialisation → XML

Introduction

- For everything you ever wanted to know about SPARQL :
- http://www.w3.org/TR/rdf-sparql-query/
  - SPARQL is a recursive acronym standing for SPARQL Protocol and RDF Query Language.
  - SPARQL queries RDF graphs. An RDF graph is a set of triples.
  - SPARQL is a general term for: a protocol + a query language + XML results.
  - SPARQL – a query language and data access protocol for the Semantic Web.
  - SPARQL is defined in terms of the W3C's RDF data model and will work for any data source that can be mapped into RDF.
  - SPARQL is a **syntactically-SQL-like language** for querying
  - RDF graphs via pattern matching.

Introduction

- Most forms of SPARQL query contain a set of triple patterns called a basic graph pattern.
- Triple patterns are like RDF triples except that each of the subject, predicate and object may be a variable.
- A basic graph pattern matches a subgraph of the RDF data when RDF terms from that subgraph may be substituted for the variables and the result is RDF graph equivalent to the subgraph.
- It is important to realize that it is the triples that matter, not the serialization.
- The serialization is just a way to write the triples down.
- RDF/XML is the W3C recommendation but it can difficult to see the triples in this serialization.

Introduction

RDF (in Turtle syntax):

```
<http://example.org/book/book1> <http://purl.org/dc/
elements/1.1/title> "SPARQL Tutorial" .
```

SPARQL Query:

```
SELECT ?title
WHERE
<http://example.org/book/book1> <http://purl.org/dc/
elements/1.1/title> ?title
```

SPARQL Query:

```
title
--------------
"SPARQL Tutorial"
```

Introduction

- SPARQL consists of three separate specifications
  - query specification language
  - query result XML format
  - data access protocol
- SPARQL consists of a query language, a means of conveying a query to a query processor service, and the XML format in which query results will be returned.

Triples

- RDF is built on the triple, a 3-tuple consisting of subject, predicate, and object.
- Likewise SPARQL is built on the triple pattern, which also consists of a subject, predicate and object.
- In fact an RDF triple is also a SPARQL triple pattern.
- A triple from our data expressed using the SPARQL triple pattern syntax looks like this

```
uol:Leicester  uol:population "328939"^^xsd:int .
```

Triples

- SPARQL specifies a number of handy abbreviations for writing complex triple patterns.
- Both the basic syntax and abbreviations borrow heavily from Turtle, a very terse RDF serialization alternative to RDF/XML.
- As a text rather than XML format, Turtle can be used to express RDF very succinctly.

Graph Pattern

- Basic Graph Patterns: set of triple patterns
- Graph Pattern: A set of graph patterns must all match.
- Value constraints: restrict RDF terms in a solution.
- Optional Graph Patterns: additional patterns may extend the solution.
- Alternative Graph Pattern: two or more possible Patterns are tried.
- Patterns on Named Graphs - Patterns are matched against named graphs

Triple patterns

- A triple pattern can include variables.
- Any or all of the subject, predicate, and object values in a triple pattern may be replaced by a variable.
- Variables are used to indicate data items of interest that will be returned by a query.
- variables: global scope, indicated by "?" or "$"

```
?p  foaf:name ?name.

#Alternative

$p  foaf:name $name.
```

Triple patterns

- This pattern will match any RDF resource that has a `foaf:knows` property.
- Each triple that matches the pattern will bind an actual value from the RDF dataset to each of the variables.
- For example,

```
?p1  foaf:name ?name.
```

- given the data below

```
<http://www.cs.le.ac.uk/rdf#johnsmith>  foaf:name  "John Smith"
```

Variable `p1` is bound to `<http://www.cs.le.ac.uk/rdf#johnsmith>` and variable `p2` is bound to string `"John Smith"`

Triple patterns: example

A basic pattern is a set of triple patterns. It matches when the triple patterns all match with the same value used each time the variable with the same name is used

Given the RDF below

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix uol: <http://www.cs.le.ac.uk/rdf#> .

uol:tom  foaf:knows uol:alice
uol:john foaf:knows uol:bob
uol:john foaf:knows uol:tim
uol:alice  uol:study  uol:CO7216
uol:bob  uol:study  uol:CO7216
```

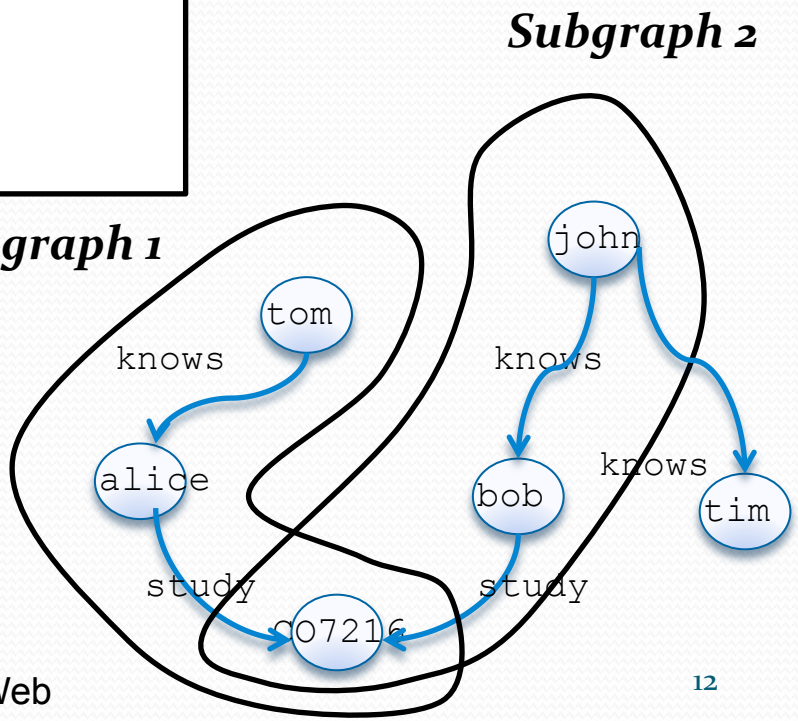
Triple pattern

```
?p1  foaf:knows ?p2.
?p2  uol:study  uol:CO7216 .
```

Result

```
p1                p2
------------------------
uol:tom     uol:alice
uol:john    uol:bob
```

For example: The triple pattern below will match *subgraph1* and *subgraph 2*



Triple patterns:

This pattern matches all triples in an RDF graph.

```
?subject ?predicate ?object
```

Results:

```
 subject      predicate     object
-----------------------------------
uol:tom       foaf:knows    uol:alice
uol:john      foaf:knows    uol:bob
uol:john      foaf:knows    uol:tim
uol:alice     uol:study     uol:CO7216
uol:bob       uol:study     uol:CO7216
```

Structure of a SELECT Query (SPARQL v1.1)

Example:

this SPARQL Query selects all persons' names

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT  ?name
FROM    <http://example.org/foaf/aliceFoaf>
WHERE   { ?x foaf:name ?name }
```

- PREFIX ..
- SELECT ..
- FROM ..
- WHERE { … }

Understanding the SELECT query

- `PREFIX`
  - shorthand mechanism for writing long URIs
  - PREFIX is essentially the SPARQL equivalent of declaring an XML namespace: it associates a short label with a specific URI.
  - A query can include any number of PREFIX statements. The label assigned to a URI can be used anywhere in a query in place of the URI itself; for example, within a triple pattern.
  - For example, In the single triple pattern included in this query we can see the table prefix in use as a shorthand for `http://xmlns.com/foaf/0.1/name` the full UIR of the `foaf:name` property

Understanding the SELECT query

- `BASE` is another form of URI abbreviation, defining the base URI against which all relative URIs in the query will be resolved, including those defined with `PREFIX`.

For example:

```
BASE <http://www.cs.le.ac.uk/rdf#>
```

Then:

```
<alice>
```

is same as

```
<http://www.cs.le.ac.uk/rdf#alice>
```

SELECT clause

- SELECT
  - Like its twin in a SQL query, the SELECT clause is used to define the data items that will be returned by a query. For example: `SELECT ?name,` we're returning a single column, the name of the person.

```
SELECT ?var1 ?var2 ..
```

Query results should include `var1` and `var2`

```
SELECT *
```

Query results should include any variables used in the graph pattern

```
SELECT DISTINCT ?var
```

Duplicated values are not included in the query results

From Clause

- FROM
  - The FROM keywords allow a query to specify an RDF dataset by reference.
  - The FROM keyword identifies the data against which the query will be run.
  - A query may actually include multiple FROM keywords, as a means to assemble larger RDF graphs for querying.
  - If there is no FROM clause then it is referring to the default graph

```
FROM <http://example.org/foaf/aliceFoaf>
FROM NAMED <http://example.org/foaf/bobFoaf>
FROM NAMED <http://example.org/foaf/tomFoaf>
```

Searching a default graph and two named graphs

Multiple triple patterns

Example:

Find somebody who knows another person who is 25 years old.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?p1
WHERE {
?p1  foaf:knows ?p2 .
?p2  foaf:age  25 .
}
```

WHERE clause

- `WHERE`
  - A graph pattern is a collection of triple patterns that identify the shape of the graph that we want to match against.
  - The `WHERE` keyword is actually optional and can legally be omitted to make queries slightly terser

```
WHERE{
Triple pattern 1.
Triple pattern 2.
…
Triple pattern n
}
```

Shortcuts

- predicate-object list, similar to that turtle syntax. It allows a query author to list the subject of a series of triple patterns only once. Similar for object lists.
- When we're using this form, each triple pattern is terminated with a semicolon rather than a full stop.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?fname ?gname
WHERE {
?p1   foaf:familyName ?fame;
      foaf:givenName   ?gname;
      foaf:age   ?age.
}
```

Optional Pattern Matching

- OPTIONAL
  - Provides the ability to query for data but not to fail query when that data does not exist. The query is using an optional part to extend the information found but to return the non-optional information anyway.

Example

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?age
WHERE {
?p1  foaf:name ?name.
OPTIONAL{ ?p1 foaf:age  ?age}
}
```

Sample result

```
  name              age
-------------------------
 "John Smith"      20
 "Jane Roe"
```

Even Jane's age is undefined, the query will still return other non-optional information.

Matching Alternatives

- UNION
  - SPARQL provides a means of combining graph patterns so that one of several alternative graph patterns may match
  - If more than one of the alternatives matches, all the possible pattern solutions are found.
  - Pattern alternatives are syntactically specified with the UNION

Example

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX uol: <http://www.cs.le.ac.uk/rdf#>
SELECT ?p1 ?p2
WHERE {
{?p1 foaf:know uol:bob} UNION {?p2 foaf:know uol:alice}
}
```

```
p1                  p2
-------------------------
uol:john
                    uol:tom
```

Filter Condition

- FILTER
  - Filter clause can be used to restrict the values in a solution.
  - Usage: `FILTER (<condition>)`

Example 1: Find people who are over 24 years old.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?p
WHERE {
?p foaf:age ?age.
FILTER (?age >= 24)
}
```

Example 2: Finds given names with an "h" or "H" in them using regular expression.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?p
WHERE {
?p foaf:givenName ?gname.
FILTER regex(?gname, "h", "i")}
```

*"i" means case-insensitive pattern match is done. For more info On regular expression, see https://www.w3.org/TR/rdf-sparql-query/#funcex-regex

Sorting

- Filter clause can be used to restrict the values in a solution.
  - ORDER BY: Put the solutions in order according to given variable. (ascending)
  - LIMIT: restrict the number of results (rows) returned.
  - OFFSET: control where the solutions start from.

Example:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?p
WHERE {
?p foaf:age ?age.
ORDER BY ?age
LIMIT 100
OFFSET 5
}
```

Aggregate Function

- `GROUP BY` keyword
- aggregate functions such as `COUNT()`, `SUM()` etc

Example:

Calculate the number of people in each year group

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT  ?age (COUNT(DISTINCT ?p) as ?person_count)
 WHERE{ ?p foaf:age ?age.}
GROUP BY ?age
```

Sample result:

```
?age         ?person_count
---------------------------
40               1
33               1
28               2
22               1
```

SPARQL Query Results XML Format

- https://www.w3.org/TR/rdf-sparql-XMLres/
- XML format for the variable binding and boolean results formats provided by the SPARQL query language for RDF

Example:

```
<?xml version="1.0"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
  <head>
    <variable name="p"/>
    <variable name="age"/>
  </head>
  <results>

    <result>
      <binding name="p">
                    <uri>http://www.cs.le.ac.uk/rdf#johnsmith</uri>
      </binding>
      <binding name="age">
              <literal datatype="http://www.w3.org/2001/XMLSchema#integer">10</literal>
      </binding>
    </result>

    ...
  </results>

</sparql>
```

SPARQL Query Results XML Format

- All of the key elements belong to a single namespace, http://www.w3.org/2005/sparql-results#
- The root element is sparql, which contains a head and a results element that together describe the result set
- The head section declares all variables that will be returned in the result set. It's equivalent to the column headings in an HTML table
- The results section lists each query result, i.e. one result element for each row in the result set
- A result element contains one binding for each variable. A binding is one of literal or uri. These elements contain the actual values returned. If a variable is not bound in a query (see the above section on OPTIONAL Patterns), then it is marked as unbound.

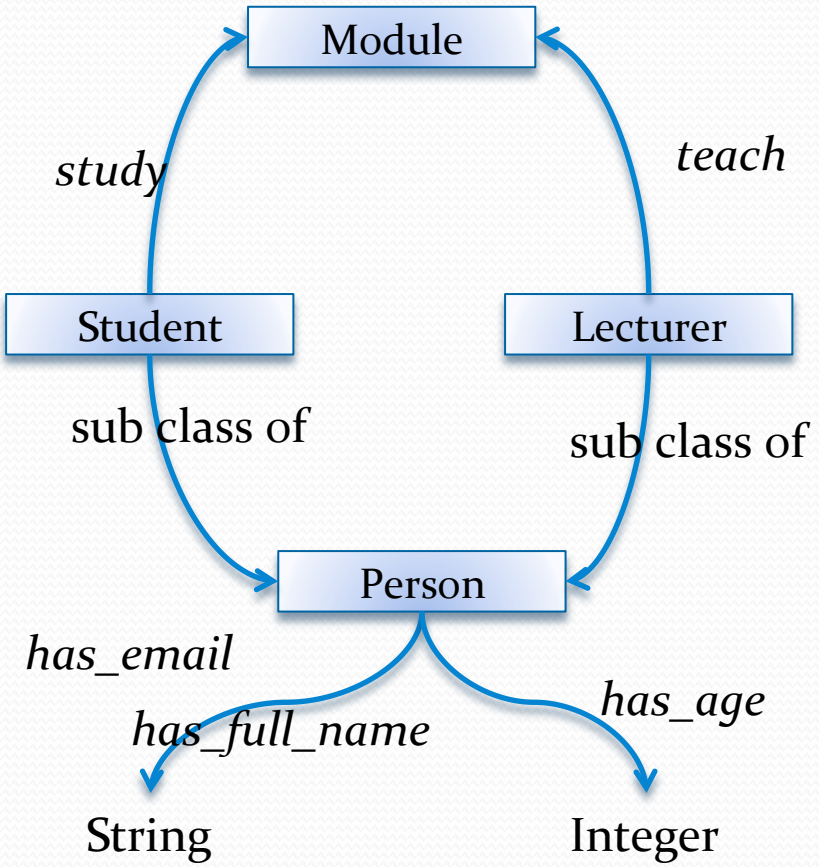
More Examples: Given a RDF below

```
@prefix rdfs:    <http://www.w3.org/2000/01/
rdf-schema#> .
@prefix :  <http://www.cs.le.ac.uk/rdf#> .
@prefix uol:     <http://www.cs.le.ac.uk/
rdfs#> .
@prefix xsd:     <http://www.w3.org/2001/
XMLSchema#> .
@prefix rdf:     <http://www.w3.org/
1999/02/22-rdf-syntax-ns#> .

:has_age a       rdf:Property ;
      rdfs:domain :Person ;
      rdfs:range xsd:int .

:s3 a       :Student ;
      :has_age "28"^^xsd:int ;
      :has_email
"s3@student.le.ac.uk"^^xsd:string ;
      :study :CO7216 , :CO7215 .

:study a       rdf:Property ;
      rdfs:domain :Student ;
      rdfs:range :Module .

:CO1003 a       :Module .

:Student a       rdfs:Class ;
      rdfs:subClassOf :Person .

:Person a rdfs:Class .
```

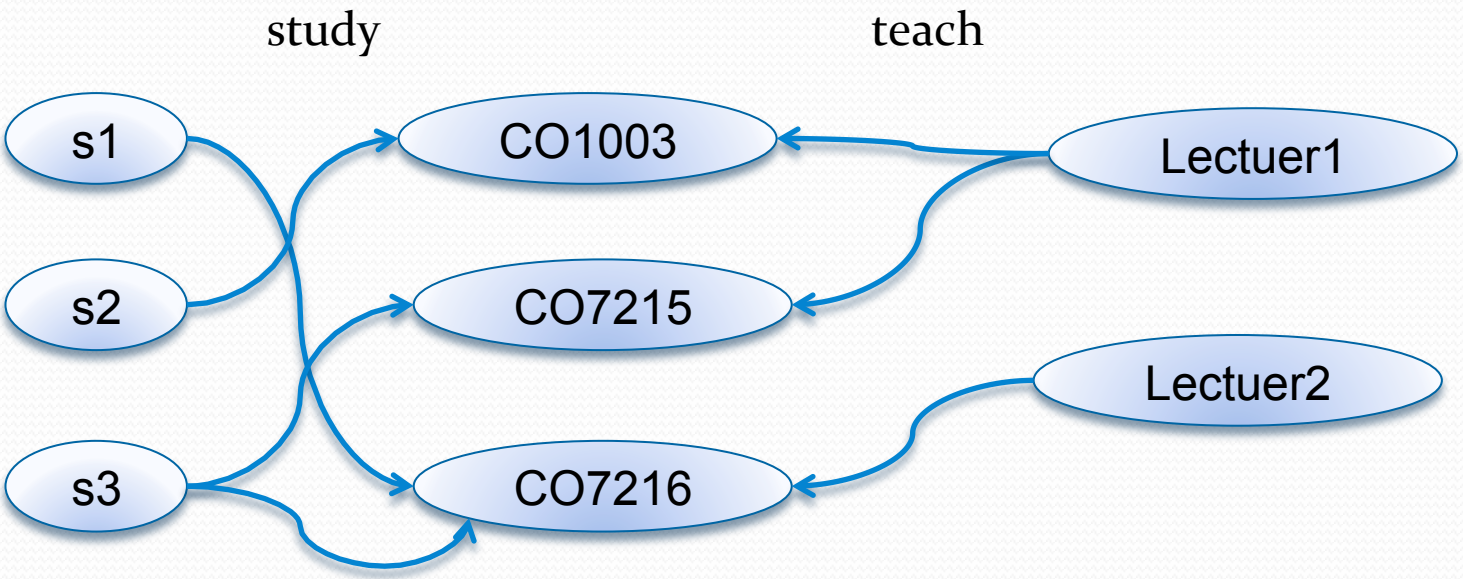
```
:teach a rdf:Property ;
      rdfs:domain :Lecturer ;
      rdfs:range :Module .

:Lecturer2 a       :Lecturer ;
      :has_age "33"^^xsd:int ;
      :has_email "l2@le.ac.uk"^^xsd:string ;
      :teach :CO7216 .

:Lecturer1 a       :Lecturer ;
      :has_age "40"^^xsd:int ;
      :has_email "l1@le.ac.uk"^^xsd:string ;
      :teach :CO1003 , :CO7215 .

:Lecturer a       rdfs:Class ;
      rdfs:subClassOf :Person .

:has_email a       rdf:Property ;
      rdfs:domain :Person ;
      rdfs:range xsd:string .

:has_full_name
      a       rdf:Property ;
      rdfs:domain :Person ;
      rdfs:range xsd:string .

:CO7216 a       :Module .
```

More Examples:

```
:Module a       rdfs:Class .

:s2 a       :Student ;
      :has_age "22"^^xsd:int ;
      :has_email
"s2@student.le.ac.uk"^^xsd:string ;
      :has_full_name
               ""^^xsd:string ;
      :study :CO1003 .

:CO7215
      a       :Module .

:s1
      a       :Student ;
      :has_age "28"^^xsd:int ;
      :has_email
"s1@student.le.ac.uk"^^xsd:string ;
      :study :CO7216 .
```



RDF Graph:


study
teach
s1
s2
s3
CO1003
CO7215
CO7216
Lectuer1
Lectuer2

Examples

Query 1: Get the students taking a module taught by L1

```
PREFIX uol: <http://www.cs.le.ac.uk/rdf#>
SELECT ?student
WHERE
{
uol:L1 uol:teach ?module.
?student uol:study ?module
}
```

Query 2: Get all lecturer who are in their 30s, display their names and emails (if given)

```
PREFIX uol: <http://www.cs.le.ac.uk/rdf#>
SELECT ?p ?age ?email
WHERE {
?p a uol:Lecturer.
?p uol:has_age ?age .
OPTIONAL  {?p uol:has_email ?email}
FILTER (?age>=30 && ?age<=40)
}
```

Examples

Query 3: Show possible relationship between `s1` and `CO7216`

```
PREFIX uol: <http://www.cs.le.ac.uk/rdf#>
SELECT ?relation
WHERE {uol:s1 ?relation uol:CO7216}
```

Query 4: Count the number of students in each module

```
PREFIX uol: <http://www.cs.le.ac.uk/rdf#>
SELECT ?module  (count(DISTINCT ?s) as ?s_count)
WHERE
{
?s a uol:Student.
?s uol:study ?module.
}
GROUP BY ?module
```

Resources

- W3C SPARQL specification https://www.w3.org/TR/sparql11-query/
- This talk inspired by: Introducing SPARQL: Querying the Semantic Web, can be found on http://www.xml.com and SPARQL Tutorial: Apache Jena https://jena.apache.org/
- SPARQLer http://sparql.org/, is an online demo of Jena's SPARQL query engine. You can use it to experiment with SPARQL queries.
- ARQ A SPARQL Processor for Jena: https://jena.apache.org/documentation/query/
- SPARQL FAQ: http://thefigtrees.net/lee/sw/sparql-faq