COMP318

SPARQL

www.csc.liv.ac.uk/~valli/Comp318

Dr Valentina Tamma

Room: Ashton 2.12

Dept of computer science

University of Liverpool

V.Tamma@liverpool.ac.uk

Where were we

- RDF and RDFS
- Vocabulary and model
- Entailment in RDF(S)
  - Simple entailment

SPARQL in general

- SPARQL Protocol and RDF Query Language
- SPARQL Query Language for RDF
  - Declarative
  - Based on the RDF data model (triples/graph)
- SPARQL Query Results XML Format
  - Representation of the results of SPARQL queries
- SPARQL Protocol for RDF
  - Transmission of SPARQL queries and the results
  - SPARQL endpoint: Web service that implements the protocol

SPARQL

- SPARQL is the query language for querying RDF. It allows users to:
  - Pull values from ***structured*** and ***semi-structured*** data
  - Explore data by querying ***unknown relationships***
  - Perform ***complex joins*** of ***disparate databases*** in a single, simple query
  - ***Transform RDF data*** from one vocabulary to another

Assumptions

- Data is represented in RDF

```
subject - predicate - object
```

- Resources are represented by URIs
  - possibly abbreviated as ***prefixed names***
- Objects can be ***literals:*** strings, integers, booleans...
- We use Turtle syntax:
  - URIs:
    - `<http://www.sw-example.com/resource>` or `prefix:name`
  - literals:
    - `“plain string” “14.1”^^xsd:float` or `”string with language”@en`
  - triples:
    - `pref:subject anOtherPref:predicate “object”`

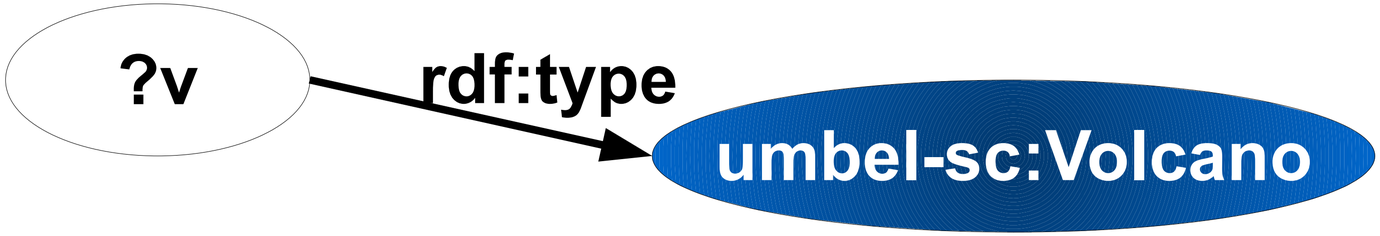
SPARQL versions

- SPARQL 1.0 (2008) included:
  - SPARQL 1.0 Query Language
  - SPARQL 1.0 Protocol
  - SPARQL Results XML Format

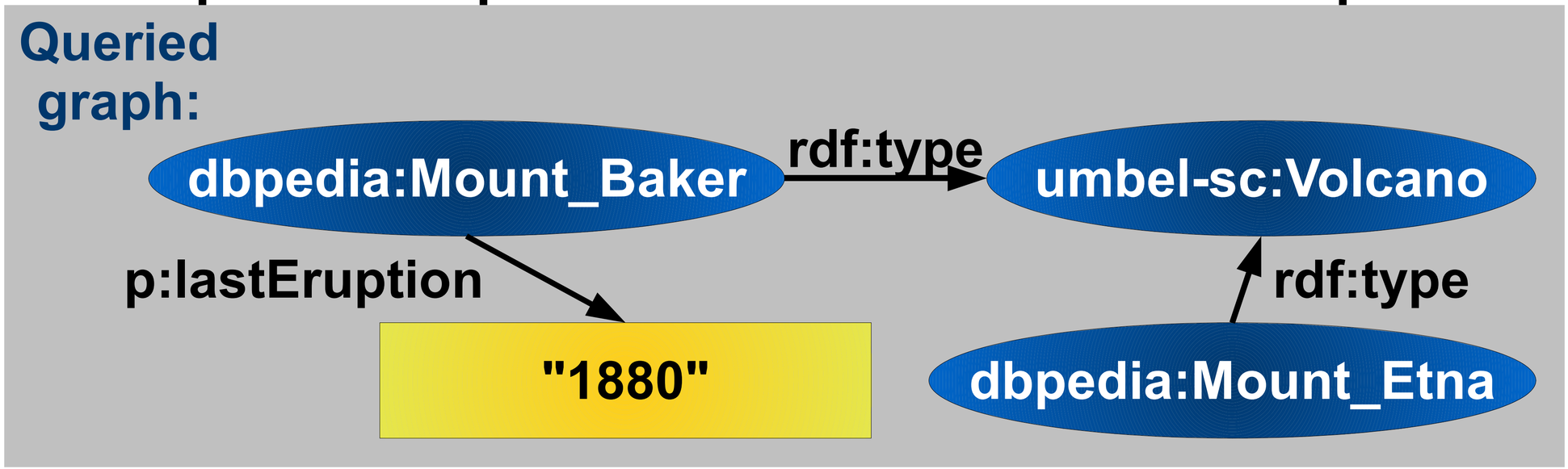
- SPARQL 1.1 (2013) includes:
  - SPARQL 1.1 versions of SPARQL Query and SPARQL Protocol
  - SPARQL 1.1 Update
  - SPARQL 1.1 Graph Store HTTP Protocol
  - SPARQL 1.1 Service Description
  - SPARQL 1.1 Entailments
  - SPARQL 1.1 Basic Federated Query

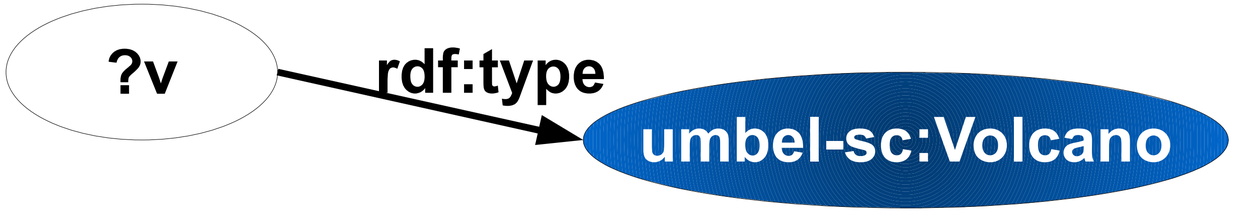
Main principle of SPARQL queries

- SPARQL based on the principle of pattern matching
  - Describe subgraphs of the queried RDF graph
  - Subgraphs that match your description yield a result
    - i.e. graph patterns (i.e. RDF graphs with variables)



Main principle of SPARQL queries





Results:

?v
dbpedia:Mount_Baker
dbpedia:Mount_Etna

SPARQL queries

- PREFIX
  - Prefix mechanism for abbreviating URIs
    - **`PREFIX foo: <http://example.com/resources/>`**
- Query Pattern
  - Result clause
    - Identifies the variables to be returned in the query answer
    - **`SELECT ...`**
- DATASET DEFINITION
  - Name(s) of the graph(s) to be queried
    - **`FROM ...`**
- QUERY PATTERN
  - Specifying what to query for in the dataset
    - Query pattern as a list of triple patterns
    - **`WHERE {`**
      `...`
      `}`
- QUERY MODIFIERS
  - **`ORDER BY ...`**

Query execution

- SPARQL queries are executed against RDF datasets (made by RDF graphs).

- A SPARQL ***endpoint*** accepts queries and returns results via HTTP.
  - endpoint is the entry point to a service, a process, or a queue or topic destination
  - ***Generic*** endpoints will query any Web-accessible RDF data
  - ***Specific*** endpoints are hardwired to query against particular datasets

SPARQL Result set

- Result sets are illustrated in:

```
x                    y                    z
=================================
"Alice"    http://example/a
```

- A 'binding' is a pair (variable, RDF term). In this result set, there are three variables: x, y and z (shown as column headers).
- Each solution is shown as one row in the body of the table.
- Here, there is a single solution
  - `x` is bound to "Alice",
  - `y` is bound to `<http://example/a>`,
  - `z` is not bound to an RDF term.

URI abbreviation: PREFIX

- Mechanism for namespace abbreviation
- Syntax:

```
PREFIX abbr: <URI>
```

- Example:

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

- Default:

```
PREFIX : <URI>
```

- Example:

```
PREFIX : <http://example.org/myOntology#>
```

URI abbreviation: PREFIX

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

- Prologue:

- Prefix definitions enable CURIEs in the query
  - CURIE: defines a generic, abbreviated syntax for expressing Uniform Resource Identifiers (URIs).
  - Abbreviated URI expressed in a compact syntax, and may be found in both XML and non-XML grammars.
  - A CURIE may be considered a datatype.

- Attention: No period (“.”) character to separate (as in N3)

Selecting variables: SELECT

- Filtering variables to return
- Variables: **`?string`**

  `?x ?title ?name`
  - variables can match any node (resource or literal) in the RDF document
- Syntax:

  `SELECT var1, ... ,varn`

  `SELECT ?x,?title`

  `SELECT *`
- Variables in **`SELECT`** are distinguished variables
- **`DISTINCT`** for disjoint results

Query Patterns: FROM

- Specifies the dataset to be queried
- Can be the default dataset
  - Then **`FROM`** can be omitted
  - **`FROM`** and **`NAMED FROM`** each with a URI to specify one or more dataset

Query Patterns: WHERE

- Graph pattern to match
  - Triple patterns are just like triples, but any part of a triple pattern can be replaced with a variable
- Set of triples:
- { (**`subject predicate object`** .)* }
  - Subject: URI, QName, Blank node, Variable
  - Predicate: URI, QName, Blank node, Variable
  - Object: URI, QName, Blank node Literal, Variable
- Example:

```
{
    _:author ex:hasName ?name .
    _:author ex:authorOf :lotr .
}
```

- Optional triples: `OPTIONAL` triple .

```
OPTIONAL :john ont:hasAge ?age
```

GRAPH PATTERNS

- Different types of graph patterns for the query pattern (WHERE clause):
  - Basic graph pattern (BGP)
  - **`Group`** graph pattern
  - **`Optional`** graph pattern
  - **`Union`** graph pattern
  - **`Graph`** graph pattern (Constraints)

Example RDF Dataset (Turtle)

```
@prefix : <http://example.org/data#> .
@prefix ont: <http://example.org/myOntology#> .
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .

:john
   vcard:FN "John Smith" ;
   vcard:N [
        vcard:Given "John" ;
        vcard:Family "Smith" ] ;
   ont:hasAge 32 ;
   ont:marriedTo :mary .
:mary
   vcard:FN "Mary Smith" ;
   vcard:N [
        vcard:Given "Mary" ;
        vcard:Family "Smith" ] ;
   ont:hasAge 29 .
```

BASIC GRAPH PATTERNS

- Set of triple patterns (i.e. RDF triples with variables)
- Variable names prefixed with “?” or “$” (e.g. ?v, $v)
- Turtle syntax (similar to N3)
  - Syntactic sugar as in N3 (e.g. property and object lists)
- Blank nodes in SPARQL queries
  - Permitted as subject and object of a triple pattern
  - Like non-selectable variables

SPARQL Queries: all full names

"Return the full names of all people in the graph"

```
PREFIX vCard:
     <http://www.w3.org/2001/vcard-rdf/3.0#>
SELECT ?fullName
WHERE {?x vCard:FN ?fullName}
```

result:

```
fullName
=================
"John Smith"
"Mary Smith"
```

```
:john
   vcard:FN "John Smith" ;
   vcard:N [
        vcard:Given "John" ;
        vcard:Family "Smith" ] ;
   ont:hasAge 32 ;
   ont:marriedTo :mary .
:mary
   vcard:FN "Mary Smith" ;
   vcard:N [
        vcard:Given "Mary" ;
        vcard:Family "Smith" ] ;
   ont:hasAge 29 .
```

SPARQL Queries: properties

"Return the relation between John and Mary"

```
PREFIX : <http://example.org/myOntology#>
SELECT ?p
WHERE { :john ?p :mary }
```

result:

```
p
==================
<http://example.org/myOntology#marriedTo>
```

SPARQL Queries: complex patterns

"Return the spouse of a person by the name of John Smith"

```
PREFIX vCard: <http://www.w3.org/2001/vcard-rdf/3.0#>
PREFIX ont: <http://example.org/myOntology#>
SELECT ?y
WHERE {?x vCard:FN "John Smith".
       ?x ont:marriedTo ?y}
```

result:

```
y
=================
<http://example.org/data#mary>
```

```
:john
    vcard:FN "John Smith" ;
    vcard:N [
          vcard:Given "John" ;
          vcard:Family "Smith" ] ;
    ont:hasAge 32 ;
    ont:marriedTo :mary .
:mary
    vcard:FN "Mary Smith" ;
    vcard:N [
          vcard:Given "Mary" ;
          vcard:Family "Smith" ] ;
    ont:hasAge 29 .
```

SPARQL Queries: blank nodes

"Return the name and the first name of all people in the KB"

```
PREFIX vCard: <http://www.w3.org/2001/vcard-rdf/3.0#>
SELECT ?name, ?firstName
WHERE {?x vCard:N ?name .
       ?name vCard:Given ?firstName}
```

result:

```
      name          firstName
=============================
_:a "John Smith"  "John"
_:b "Mary Smith"  "Mary"
```

OPTIONAL GRAPH PATTERNS

- Used to treat missing data
- Keyword **`OPTIONAL`** allows for optional patterns
- May yield unbound variables

SPARQL Queries: optional pattern

"Return the full name of all the people in the graph and their spouse"

```
PREFIX vCard: <http://www.w3.org/2001/vcard-rdf/3.0#>
PREFIX ont: <http://example.org/myOntology#>
SELECT ?y ?name
WHERE {?x vCard:FN ?name.
    OPTIONAL {?x ont:marriedTo ?y}}
```

```
:john
    vcard:FN "John Smith" ;
    vcard:N [
          vcard:Given "John" ;
          vcard:Family "Smith" ] ;
    ont:hasAge 32 ;
    ont:marriedTo :mary .
:mary
    vcard:FN "Mary Smith" ;
    vcard:N [
          vcard:Given "Mary" ;
          vcard:Family "Smith" ] ;
    ont:hasAge 29 .
```

SPARQL Queries: optional pattern

"Return the full name of all the people in the graph and their spouse"

```
PREFIX vCard: <http://www.w3.org/2001/vcard-rdf/3.0#>
PREFIX ont: <http://example.org/myOntology#>
SELECT ?y ?name
WHERE {?x vCard:FN ?name.
   OPTIONAL {?x ont:marriedTo ?y}}
results:
?name                          ?y
================================
"John Smith"  <http://example.org/data#mary>
"Mary Smith"
```