Information Visualization

Data access and query

Lesson 4

Marilena Daquino
Assistant Professor

Department of
Classical Philology
and Italian Studies

marilena.daquino2@unibo.it

Table of contents

01

SPARQL language

The standard query language for querying RDF data.

SPARQL query types

01 SELECT

Get a partial view of data. Returns a SPARQL result.

02 CONSTRUCT

Build a new graph out of an existing one. Returns a graph

03 ASK

Returns whether information exist or not. Returns a SPARQL result.

04 INSERT / DELETE

Update and returns a graph.

05 DESCRIBE

Returns the graph of an entity

SELECT query

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

SELECT ?person

WHERE {
     ?person rdf:type wd:Q5 .
}
```

Give me all the URIs of individuals defined as instances of the class wd:Q5 (Human).

SELECT variables

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

```
SELECT ?person
```

```
WHERE {
     ?person rdf:type wd:Q5 .
}
```

The SELECT statements includes the **dependent variables** to be retrieved by the query. All variables are preceded by the placeholder “?”

WHERE clause

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

```
SELECT ?person
```

```
WHERE {
     ?person rdf:type wd:Q5 .
}
```

The WHERE clause specifies how to retrieve the variables. It includes a number of **triple patterns** that helps the SPARQL engine to traverse the graph. Everything is enclosed in “{}”

PREFIXES

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

```
SELECT ?person
```

```
WHERE {
     ?person rdf:type wd:Q5 .
}
```

Namespaces can be defined to simplify triple patterns in the WHERE clause.

SPARQL Results

person
http://example.org/person/1
http://example.org/person/2

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX wd: <http://www.wikidata.org/entity/>

SELECT ?person

WHERE {
      ?person rdf:type wd:Q5 .
}
```

SPARQL results can be **tabulated**. Columns correspond to the name of variables.

SELECT variables (again)

person	name
http://example.org/person/1	Federico Zeri
http://example.org/person/2	Aby Warburg

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX wd: <http://www.wikidata.org/entity/>

SELECT ?person ?name

WHERE {
      ?person rdf:type wd:Q5 ;
            rdfs:label ?name .
}
```

Multiple variables must be **dependent** (with each others). Multiple triple patterns sharing a subject can be shortened with “;”.

ORDER results

person	name
http://example.org/person/2	Aby Warburg
http://example.org/person/1	Federico Zeri

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX wd: <http://www.wikidata.org/entity/>

SELECT ?person ?name

WHERE {
      ?person rdf:type wd:Q5 ;
            rdfs:label ?name .
}
ORDER BY ?name
```

Results can be sorted alphabetically by one or more variables

LIMIT results

SPARQL

person	name
http://example.org/person/2	Aby Warburg

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX wd: <http://www.wikidata.org/entity/>

SELECT ?person ?name

WHERE {
      ?person rdf:type wd:Q5 ;
            rdfs:label ?name .
}

ORDER BY ?name
LIMIT 1
```

Results can be sorted alphabetically by one or more variables

MISSING results

SPARQL

person	name	birth
http://example.org/person/1	Federico Zeri	1920-08-12

There is no birth date for Aby Warburg in the dataset!

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX wd: <http://www.wikidata.org/entity/>
PREFIX wdt: <http://www.wikidata.org/prop/direct/>

SELECT ?person ?name ?birth

WHERE {
      ?person    rdf:type wd:Q5 ;
                 rdfs:label ?name ;
                 wdt:P569 ?birth .
}
```

A result that does not comply with **all patterns** (e.g. people without birth date) is pruned.

OPTIONAL patterns

person	name	birth
http://example.org/person/1	Federico Zeri	1920-08-12
http://example.org/person/2	Aby Warburg	

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX wd: <http://www.wikidata.org/entity/> .
PREFIX wdt: <http://www.wikidata.org/prop/direct/>

SELECT ?person ?name

WHERE {
      ?person    rdf:type wd:Q5 ;
                 rdfs:label ?name ;
OPTIONAL { ?person wdt:P569 ?birth_date . }
}
```

A **partial result** can appear if some triple patterns are OPTIONAL.

FILTER results

person	name	birth
http://example.org/person/1	Federico Zeri	1920-08-12

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX wd: <http://www.wikidata.org/entity/> .
PREFIX wdt: <http://www.wikidata.org/prop/direct/>

SELECT ?person ?name ?birth

WHERE {
      ?person    rdf:type wd:Q5 ;
                 rdfs:label ?name ;
                 wdt:P569 ?birth .

      FILTER(?birth_date >= "1920-01-01"^^xsd:date)
}
```

Rules can be applied to **filter out** results (e.g. people born after 1920)

Exploratory queries

Get to know your data

- Find the dataset documentation
- Find the vocabularies docum.

If not enough, perform a number of exploratory queries.

We will perform some queries on the SPARQL endpoint of ARTchives

LIST classes

SPARQL

class_uri
http://www.wikidata.org/entity/Q31855
http://www.wikidata.org/entity/Q5
http://www.wikidata.org/entity/Q9388534

```
SELECT DISTINCT ?class_uri

WHERE {
    ?anything a ?class_uri .
}
```

In ARTchives, classes from Wikidata are directly reused. Labels of classes are not stored though (you have a summary table here).

COUNT individuals of a class

SPARQL

count
27

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX wd: <http://www.wikidata.org/entity/>

SELECT (COUNT(DISTINCT ?person) AS ?count)

WHERE {
      ?person rdf:type wd:Q5 .
}
```

COUNT is an operator that works in the SELECT clause. DISTINCT prunes duplicate results (i.e. creates a set of unique values)

COUNT individuals of classes

SPARQL

class	count
http://www.wikidata.org/entity/Q31855	8
http://www.wikidata.org/entity/Q5	27
http://www.wikidata.org/entity/Q9388534	28

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

SELECT DISTINCT ?class (COUNT(DISTINCT ?anything) AS ?count)

WHERE {
      ?anything rdf:type ?class .
}

GROUP BY ?class ?count
```

If the SELECT does not return only one counting, results **must be grouped** by the variables (list them in the order they appear in the SELECT clause).

SAMPLE results

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT DISTINCT ?class_uri ?individual (sample(?individual_label) as ?label)

WHERE {
    ?individual a ?class_uri .
    ?individual rdfs:label ?individual_label .
}
GROUP BY ?label ?individual ?class_uri
ORDER BY ?class_uri ?label
LIMIT 10
```

DISTINCT does not prevent multiple results to show up in case, for instance, multiple labels are associated to a person. In case we want to **return only one possible value of a property**, we can use SAMPLE. Results must be grouped.

Bind VALUES

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX wd: <http://www.wikidata.org/entity/>
#PREFIX wdt: <http://www.wikidata.org/entity/> the correct URI
PREFIX wdt: <http://www.wikidata.org/wiki/Property:>  # the wrong URI in ARTchives

SELECT DISTINCT ?person (SAMPLE(DISTINCT ?plabel) AS ?person_label)
WHERE {

     VALUES ?occupation_uri { wd:Q1126160 wd:Q2994387 } # connoisseur and advisor

     ?person wdt:P106 ?occupation_uri ; rdfs:label ?plabel .
}
GROUP BY ?person ?person_label
```

You can **bind a variable to one or more VALUES**, e.g. to retrieve all people that are both connoisseurs and advisors.

02

Data integration

Mash up, dumping, federated queries and integration

Linked Open Data

Data integration

Different datasets may address information relevant to the same entities (e.g. art historians in ARTchives and Wikidata).

Source may be incomplete and can support each other in **enriching** a data source (e.g. including in ARTchives birth dates recorded in Wikidata)

Usually, data across datasets must be **reconciled** (i.e. asserting two individuals from different sources are the same entity).

In ARTchives we directly reuse Wikidata URIs, so we do not need to do that!

Linked Open Data

There are two strategies to integrate data (after reconciliation):

- **Federation.** Use Federated SPARQL query to integrate data on the fly between two or more endpoints
- **Dump and merge.** Download data from one source and store them into another data source (i.e. graph)

Federated queries

Federated queries allow you to perform a SPARQL query from an endpoint X to an endpoint Y, and to create a new graph.

Results of the query can include data from both the endpoints.

Federation is possible if both the endpoints are **CORS-enabled**.

Cross-Origin Resource Sharing (CORS) *is a mechanism that allows a web application running at one origin, access to selected resources from a different origin.*

Federated queries

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT ?person ?birth_place
WHERE {
    SERVICE <https://dbpedia.org/sparql/> {
    # Federico Zeri
    ?person owl:sameAs <http://www.wikidata.org/entity/Q1089074> ;
        <http://dbpedia.org/ontology/birthPlace> ?birth_place .
    }
}
```

Try it on ARTchives SPARQL endpoint

Dump and merge

Alternatively, one may perform a SPARQL query to an endpoint, download results, parse them and include them into another graph.

Cross-Origin Resource Sharing (CORS) *is a mechanism that allows a web application running at one origin, access to selected resources from a different origin.*

Dump and merge

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT ?person ?birth_place
WHERE {
    ?person owl:sameAs <http://www.wikidata.org/entity/Q1089074> ;
        <http://dbpedia.org/ontology/birthPlace> ?birth_place .
    }
}
```

Try it on DBPedia SPARQL endpoint

03

Hands-on

Jupyter notebook, SPARQLWrapper

Hands-on

Get all the materials

If you haven’t done it yet…

Download the data (resources/artchives.nq) in a folder for the exercise

Tutorial

Open the tutorial: in GitHub, Colab or Jupyter (download)

Install packages

In the terminal/shell (if IDE or Jupyter)

```
pip install SPARQLWrapper
```

Practice

Choose your environment:
IDE: create a .py file
Jup: create .ipynb file
Colab: new notebook

Exercise

Assignment

Review

Review the tutorial

Exercise

Solve the problems (time to code!)

Fill in the form with your answers

TODO

Come prepared! Install these libraries

```
pip install pandas
pip install pandas_profiling
pip install seaborn
```

Thanks!

Do you have any questions?

marilena.daquino2@unibo.it

https://github.com/marilenadaquino/information_visualization