ELECTRONIC PUBLISHING
AND DIGITAL STORYTELLING
Lesson 4

DATA ACCESS AND QUERY

TABLE OF CONTENTS

01 SPARQL LANGUAGE

02 ACCESS DATA REMOTELY

03 DATA INTEGRATION

04 HANDS-ON

ELECTRONIC PUBLISHING
AND DIGITAL STORYTELLING
Lesson 5

SPARQL

SPARQL LANGUAGE

SPARQL is the standard language for querying RDF data.

Like RDF, the serialisation of data to be queried does not matter, while **the content of data** matters, since you need to know how your data are organised in order to query them appropriately and get correct answers.

HOW DOES IT WORK

You can query data and get a partial view of those by using a **SELECT** query.

You can understand whether some data are in the dataset by using a **ASK** query.

You may want to build on top of existing data new triples by using a **CONSTRUCT** query and get enhanced results.

You can UPDATE the data you are querying, by either **INSERT** or **DELETE** queries.

A SELECT QUERY

It is made up of 2 main components:

SELECT statement: includes the dependant variables to be returned by the query with **?** as a marker.

The **WHERE** clause where you specify how data should be found. It includes a number of triple patterns that point to the selected variables. Everything is enclosed in curly brackets **{}**.

Give me all the URIs of entities that belong to the class Q5 (people).

```
SELECT ?person

WHERE {
        ?person <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.wikidata.org/entity/Q5> .
}
```

A 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 .
}
```

You can specify **PREFIXES** of namespaces in order to make the query more readable.

A 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 .
}
```

Returns

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

The results of a query can be **tabulated**, and will have as many columns as the number of selected query variables.

A SELECT QUERY

```
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 .
}
```

More variables can be selected, as long as these are **dependent** and the resulting table makes sense.

Multiple patterns sharing the same subject can be shorten by using the semicolon ;

Returns

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

Give me all the URIs and the labels of entities that belong to the class Q5.

A SELECT QUERY

```
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 **ordered** according to a query variable.

Returns

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

Give me all the URIs and the labels of entities that belong to the class Q5 ordered (alphabetically) by name.

A SELECT QUERY

```
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 .
}
LIMIT 2
```

Returns

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

The number of results can be **limited**.

Give me 2 URIs and the labels of entities that belong to the class Q5 ordered (alphabetically) by name.

A SELECT QUERY

```
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 ;
              wdt:P569 ?birth_date .

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

Returns

person	**name**
http://example.org/person/1	Federico Zeri

Results can be **filtered** according to rules specified in the WHERE clause.

Give me all the URIs and the labels of entities that belong to the class Q5 that have as value of prop. wdt:569 a value > 1920 .

A SELECT QUERY

```
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_date

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

Returns

person	**name**	**birth_date**
http://example.org/person/1	Federico Zeri	1921-08-12

Sometimes datasets lack of some information (e.g. birthdates). When specifying a pattern in the WHERE clause this is deemed mandatory. Hence, if an entity does not respect the pattern (e.g. does not have a birth date recorded) this is excluded from results.

Give me all the URIs and the labels of entities that belong to the class Q5 that have SOME value for the properties rdf:type, rdfs:label and wdt:569.

A SELECT QUERY

```
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_date

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

Returns

person	**name**	**birth_date**
http://example.org/person/1	Federico Zeri	1921-08-12
http://example.org/person/2	Aby Warburg	

We can use, inside the WHERE clause, **OPTIONAL** clauses to specify which pattern are not mandatory, but that we would like to return if available.

For missing variables, an empty cell is returned in results.

Give me all the URIs and the labels of entities that belong to the class Q5 and that MAY HAVE SOME value for the property wdt:569.

GET TO KNOW YOUR DATA

In order to perform correct and efficient SPARQL queries, you need to know how data are organised (e.g. what is the property for birth dates? How do I unequivocally distinguish historians from other people?)

1. Look for the dataset documentation if it includes mention of the vocabularies used and how these are used.
2. Look for the vocabularies documentation if you have doubts.
3. If the previous ways fail, you may have to perform **exploratory SPARQL queries** on your data (which is our case in ARTchives!).

ELECTRONIC PUBLISHING
AND DIGITAL STORYTELLING
Lesson 5

ACCESS DATA REMOTELY

DATA ACCESS METHODS ON THE WEB

Data can be served on the web in several ways.

Method	Result	CONS
Search and browsing interfaces	Paginated results in a web browser, eventually with export options.	Query capabilities limited to providers’ design choices.
Data dumps	Static file to be queried locally.	Data are not updated, and may require to be updated often.
REST APIs	Systematic access to live data.	**Requires web and coding skills.**

WEB APIs

An API (Application Programming Interface) is a middleware application that allows developers to access data served online according to a specific protocol.

Web APIs are **request-response** services to share / manipulate data via a **service URL**.
They adopt the **HTTP** protocol for the exchange and they provide data in one or more **format** (e.g. JSON, XML, CSV).

EXAMPLE: The API of the MEtropolitan Museum
DOCUMENTATION: https://metmuseum.github.io/

REQUEST: https://collectionapi.metmuseum.org/public/collection/v1/objects/15003
RETURN: a JSON file including information on the specified object 15003.

API KEYS

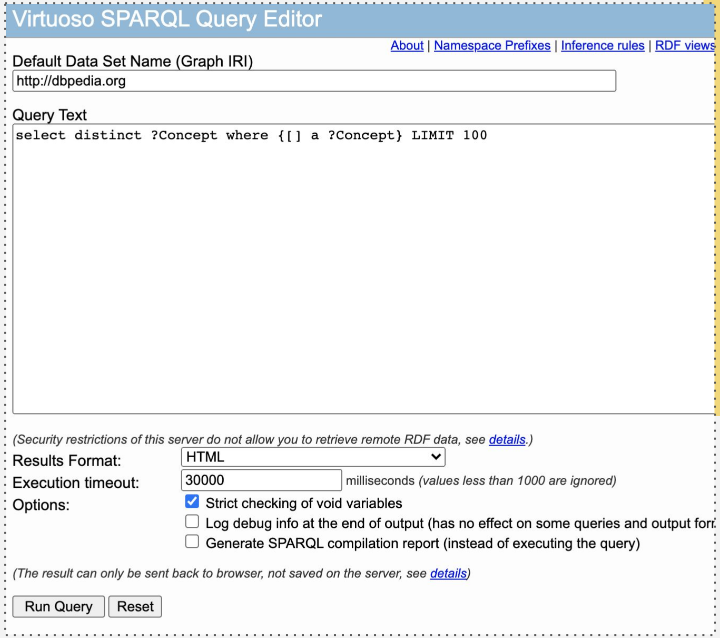
Sometimes providers need to track usage of the APIs, to authenticate users and provide different services (eventually for charge), and to limit the number of requests.

To do so, developers are quested to register for an API key, that is usually passed to request url as a **parameter**.

```
(FAKE) EXAMPLE
API KEY: abcde1986
REQUEST: https://collectionapi.metmuseum.org/public/collection/v1/objects/15003?key=abcde1986
RETURN: a JSON file including information on the specified object 15003.
```

ACCESS RDF DATA REMOTELY

DBPedia SPARQL endpoint Query interface
https://dbpedia.org/sparql



Virtuoso SPARQL Query Editor

Default Data Set Name (Graph IRI)
http://dbpedia.org

Query Text

```
select distinct ?Concept where {[] a ?Concept} LIMIT 100
```

(Security restrictions of this server do not allow you to retrieve remote RDF data, see details.)

Results Format: HTML

Execution timeout: 30000 milliseconds *(values less than 1000 are ignored)*

Options:
- ☑ Strict checking of void variables
- □ Log debug info at the end of output (has no effect on some queries and output for
- □ Generate SPARQL compilation report (instead of executing the query)

(The result can only be sent back to browser, not saved on the server, see details)

Usually RDF data served on the web are stored in dedicated graph databases (also called **triplestores**).

Such databases provide a SPARQL endpoint, that is, a service for querying data by means of SPARQL language.

Data providers may offer “user-friendly” interfaces for querying data in SPARQL. Results can be downloaded in several formats.

ACCESS RDF DATA REMOTELY

DBPedia SPARQL endpoint Query interface
https://dbpedia.org/sparql

EXAMPLE QUERY via API:
http://dbpedia.org/sparql?default-graph-uri=http%3A%2F%2Fdbpedia.org&query=SELECT+DISTINCT+%3Fp%0D%0AWHERE%0D%0A++%7B+%0D%0A++++OPTIONAL+%7B+%3Chttp%3A%2F%2Fdbpedia.org%2Fresource%2FIBM%3E++%3Fp++%3Fo++++++++++++++++++++++++++++++++%7D%0D%0A++++OPTIONAL+%7B++++++++++++++++++++++++++++++++%3Fs++%3Fp++%3Chttp%3A%2F%2Fdbpedia.org%2Fresource%2FIBM%3E+%7D%0D%0A++%7D%0D%0AORDER+BY+%3Fp&format=text%2Fx-html%2Btr&CXML_redir_for_subjs=121&CXML_redir_for_hrefs=&timeout=30000&debug=on&run=+Run+Query+

Try to copy & paste this long URL here:
https://www.url-encode-decode.com/
Press the button DECODE URL

A SPARQL endpoint also provides APIs, which allow to perform queries remotely by passing an **encoded** SPARQL query as a parameter in the URL.

This allows us to request always up-to-date data.

ELECTRONIC PUBLISHING
AND DIGITAL STORYTELLING
Lesson 5

DATA INTEGRATION

LINKED OPEN DATA

Linked Open Data are RDF data that include **links to external datasets**.

For instance, in the case of ARTchives, many of the URIs identifying art historians are directly taken and reused from Wikidata.

The strength of this type of data is that they allow to **integrate information** belonging to different sources and get a greater view of information.

For instance, we saw that in ARTchives we have relations between historians and their collections (missing in wikidata), but we don’t have historians’ birth places, which are instead in wikidata.

INTEGRATE LINKED OPEN DATA

The are two main ways to integrate linked open data.

1. Using **SPARQL federated queries**
2. Harvesting data from one source and **dumping/merging data** in another dataset.

FEDERATED SPARQL QUERIES

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.

TRY IT ON ARTchives endpoint:
http://artchives.fondazionezeri.unibo.it/sparql

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

SELECT ?person ?birth_place
WHERE {

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

If **CORS** are enabled, it is possible from one SPARQL endpoint to query **other** SPARQL endpoints.

For instance we can query **DBpedia** from ARTchives SPARQL endpoint in order to retrieve Federico Zeri’s birth place.

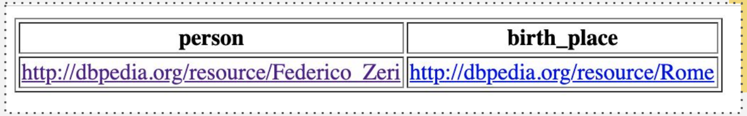
DUMPING / MERGING

On DBpedia endpoint:
https://dbpedia.org/sparql

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

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

Dump the results from DBpedia



person	birth_place
http://dbpedia.org/resource/Federico_Zeri	http://dbpedia.org/resource/Rome

Include the new triple in the artchives dataset (we will see how!)

```
<http://www.wikidata.org/entity/Q1089074> <http://www.wikidata.org/prop/direct/P569> <http:dbpedia.org/resource/Rome> .
```

We can alternatively perform queries on an external endpoint/API and merge results of the queries with our data.

ELECTRONIC PUBLISHING
AND DIGITAL STORYTELLING
Lesson 5

HANDS-ON

GET READY WITH PYTHON PACKAGES

Install the package for querying SPARQL endpoints

```
pip install sparqlwrapper
```

GET READY WITH CODING ENVIRONMENT

Open your editor and create a new python file. Use the same folder of prior tutorial:

```
dhdk_epds/
    tutorials/
        sparql_tutorial.py
```

MOVE TO THE TUTORIAL

Open the course repository on the browser
https://github.com/marilenadaquino/epds
Go to the folder tutorials/
and open in the browser the file called
sparql_tutorial.ipynb

HOMEWORK

Fill in the questionnaire! https://forms.gle/GtyReH2gfnWd38rQA

Guess what? You’ll have to code in order to answer the questions!

THANKS

Does anyone have any questions?

marilena.daquino2@unibo.it
github