SPARQL

Angelica Lo Duca
IIT-CNR
angelica.loduca@iit.cnr.it

Linked Open Data:
a paradigm for the
Semantic Web

SPARQL

- SPARQL is a query language designed specifically to query RDF databases
- SPARQL means **pattern matching**
    - find a subgraph that matches a query

Structure of a SPARQL query

```
PREFIX ex: <http://example.com/resources/>

QUERY_TYPE  Projection
FROM
WHERE {
    ...
}
QUERY MODIFIERS ...
```

Prefix

- keeps queries readable

- Examples:
  - PREFIX : <http://example.com/base/>
  - REFIX foaf: <http://xmlns.com/foaf/0.1/>

Query Types

- SELECT
    - returns a result table
- ASK
    - returns (boolean) true, if the pattern can be matched
- CONSTRUCT
    - creates triples using templates
- DESCRIBE
    - returns descriptions of resources

From Clause

- Specifies which graphs should be considered by the endpoint.
    - if omitted, the default graph is used.
    - if specified, the query is evaluated using all specified graphs.
    - if specified as named graph, the named graphs can be used in parts of the query.

Query Modifiers

- Change the result of a query
- **LIMIT** and **OFFSET** slice the result set
  - example: SELECT * WHERE {.....} LIMIT 10
    - display only 10 results

- **ORDER BY** sorts the result set
  - example: SELECT * WHERE {.....} ORDER BY ASC(...)
    - display the sorted result set

Where Clause

- Specifies the conditions

SPARQL Syntax

```
SELECT ?subject ?predicate ?object

WHERE {

   ?subject ?predicate ?object .

}
```

Search all the possible triples within the Knowledge Base

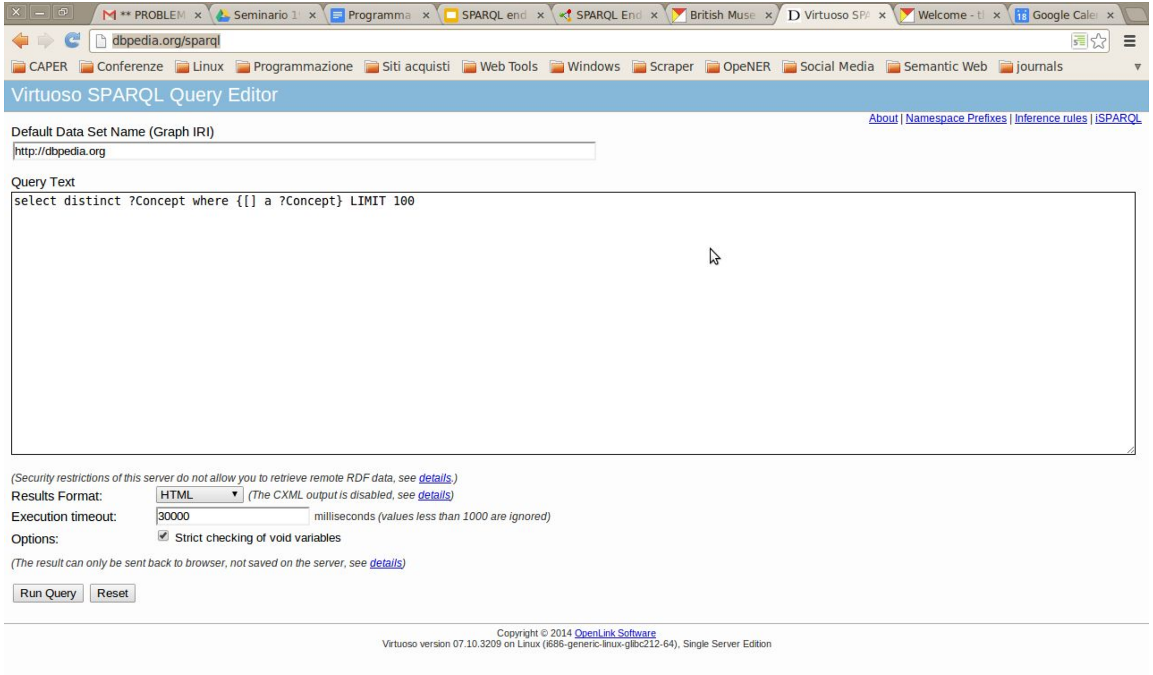
SPARQL First Example

```
PREFIX ex: <http://www.ex.com>

SELECT ?subject

WHERE {

    ?subject rdf:type ex:Animal .

}
```

Search all the possible Animals

SPARQL endpoint

- It is a service that accepts query 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 *(The CXML output is disabled, see details)*

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

Options: ☑ Strict checking of void variables

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

Copyright © 2014 OpenLink Software
Virtuoso version 07.10.3209 on Linux (i686-generic-linux-glibc212-64), Single Server Edition

DBpedia

- DBpedia is the Linked Data version of Wikipedia
- SPARQL endpoint
  - http://dbpedia.org/sparql

Search the Galileo Galilei’s birth place

- Galileo Galilei on DBpedia
  - http://dbpedia.org/resource/Galileo_Galilei
- Birth place property
  - dbo:birthPlace

Solution

```
PREFIX : <http://dbpedia.org/resource/>

SELECT ?birthPlace

WHERE {

    :Galileo_Galilei dbo:birthPlace ?birthPlace .

}
```

Syntax

```
PREFIX : <http://dbpedia.org/resource/>

SELECT ?birthdate ?deathdate
WHERE {
    :Galileo_Galilei dbp:birthDate ?birthdate .
    :Galileo_Galilei dbp:deathDate ?deathdate .
}
```

Syntax (2)

```
PREFIX : <http://dbpedia.org/resource/>

SELECT ?birthdate ?deathdate
WHERE {
    :Galileo_Galilei dbp:birthDate ?birthdate;
                     dbp:deathDate ?deathdate.
}
```

ORDER BY Operator

```
PREFIX : <http://dbpedia.org/resource/>


SELECT ?person ?birthdate
WHERE {
    ?person   dbp:birthPlace :Pisa;
              dbp:birthDate ?birthdate.
}
ORDER BY ASC(?person)
```

Select the birth date of all persons born in Pisa and order by person

FILTER Operator

Select the birth date of all persons born in Pisa between 1500-01-01 and 1900-01-01 and order by person

```
PREFIX : <http://dbpedia.org/resource/>

SELECT ?person ?birthdate
WHERE {
            ?person dbp:birthPlace :Pisa;
                        dbp:birthDate ?birthdate.
            FILTER ( ?birthdate >= '1500-01-01'^^xsd:date &&
                         ?birthdate < '1900-01-01'^^xsd:date )
}
ORDER BY ASC(?birthdate)
```

Filter Operations

- Logical: !, &&, ||
- Math: +, -, *, /
- Comparison: =, !=, >, <, ...
- SPARQL tests: isURI, isBlank, isLiteral, bound
- SPARQL accessors: str, lang, datatype, sameTerm, langMatches, regex

UNION Operator

Select the birth date of all persons born in Pisa or born in Milan

```
PREFIX : <http://dbpedia.org/resource/>

SELECT ?person ?birthdate
WHERE {
    {   ?person dbp:birthPlace :Pisa;
                dbp:birthDate ?birthdate.    }
    UNION
    {   ?person dbp:birthPlace :Milan;
                dbp:birthDate ?birthdate.}
}
```

OPTIONAL Operator

- The previous queries return only resources where the property searched is present
  - for example, if for a resource only the birth date is present but not the birth place, the resource is discarded from the query
- Optional operator overcomes this problem

OPTIONAL Operator (2)

Select persons born in Pisa. If available, select also their birth date

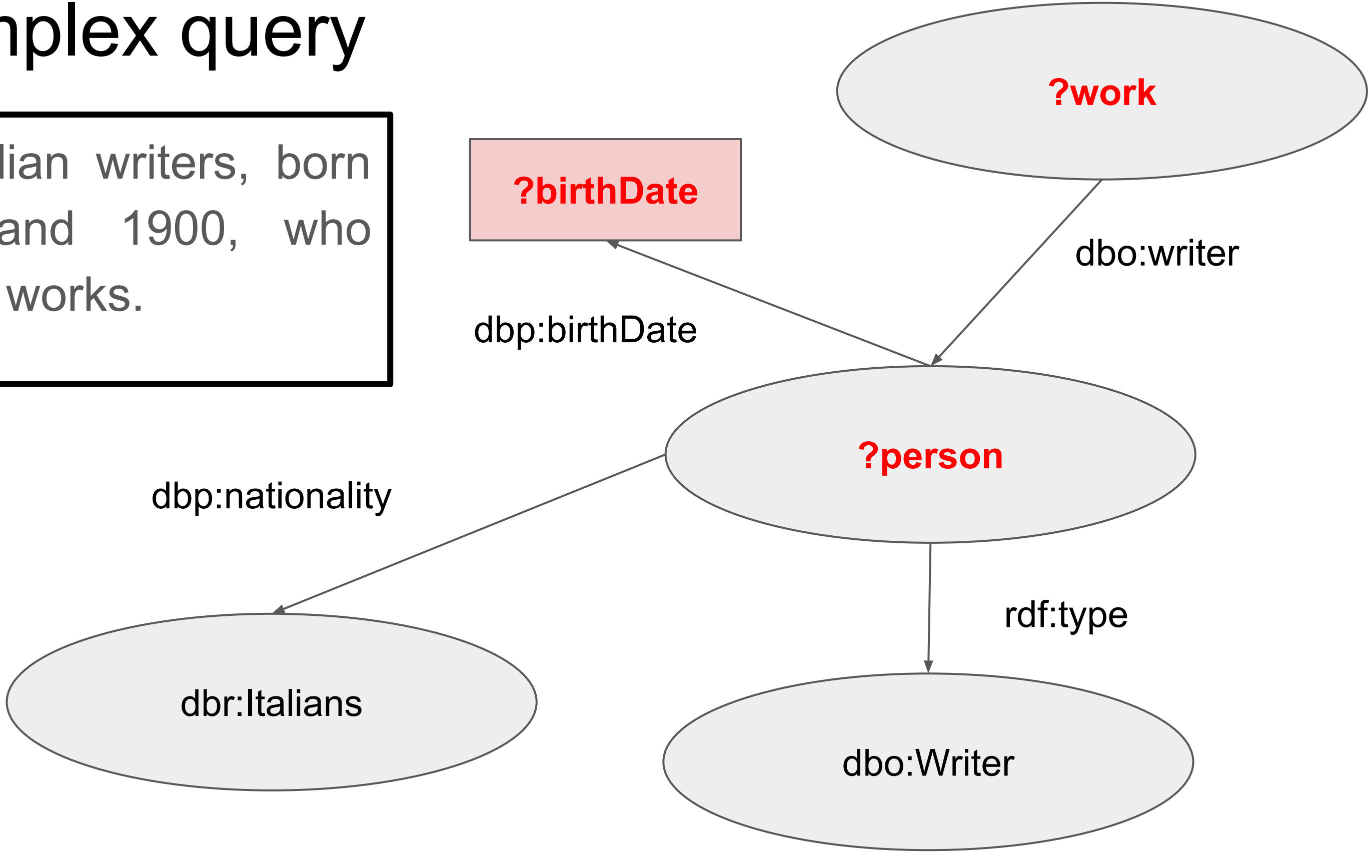
```
PREFIX : <http://dbpedia.org/resource/>

SELECT ?person ?birthDate

WHERE{

    ?person dbp:birthPlace :Pisa .

    OPTIONAL{

        ?person dbp:birthDate ?birthDate .

    }

}
```

GROUP BY - HAVING

- Works like in SQL
- In order to calculate aggregate values for a solution, the solution is first divided into one or more groups, and the aggregate value is calculated for each group
- HAVING operates over grouped solution sets,
    - FILTER operates over un-grouped ones.

A more complex query

Select all the Italian writers, born between 1500 and 1900, who wrote at least two works.



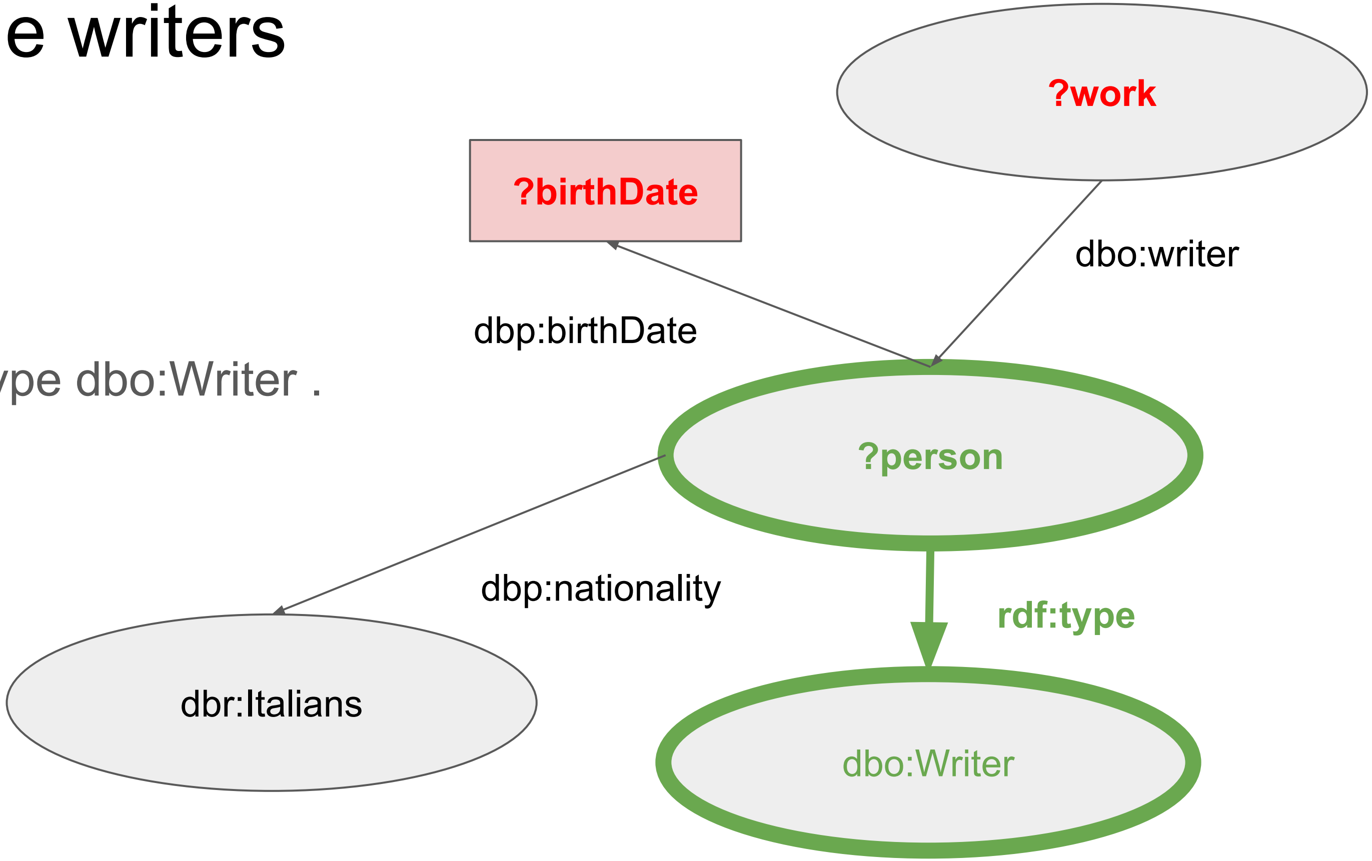
Note on DBpedia prefixes

http://dbpedia.org/sparql

- dbr - dbpedia resources
- dbo - dbpedia ontology
- dbp - dbpedia properties

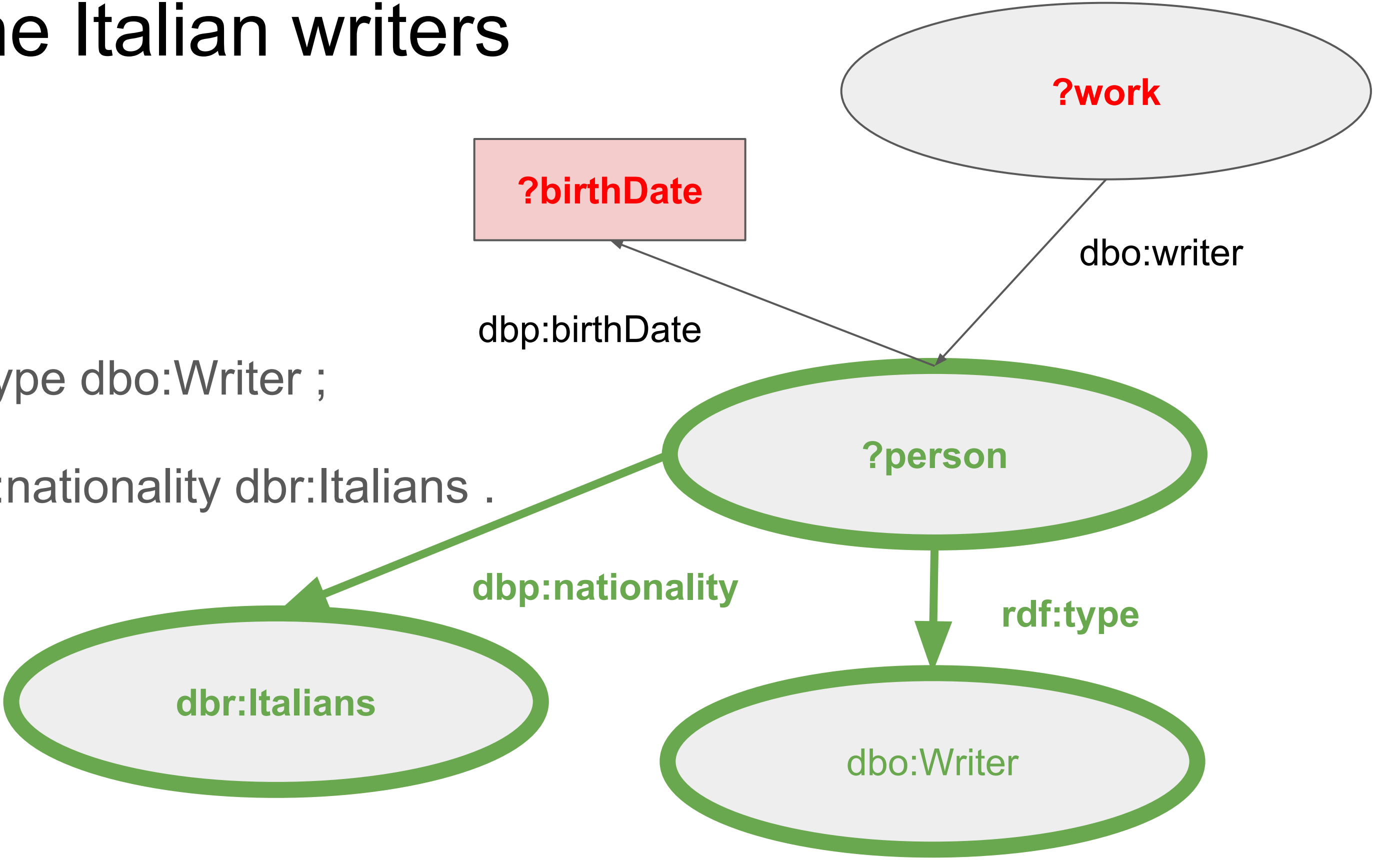
Select all the writers

```
SELECT ?person

WHERE{

      ?person rdf:type dbo:Writer .

}

LIMIT 10
```



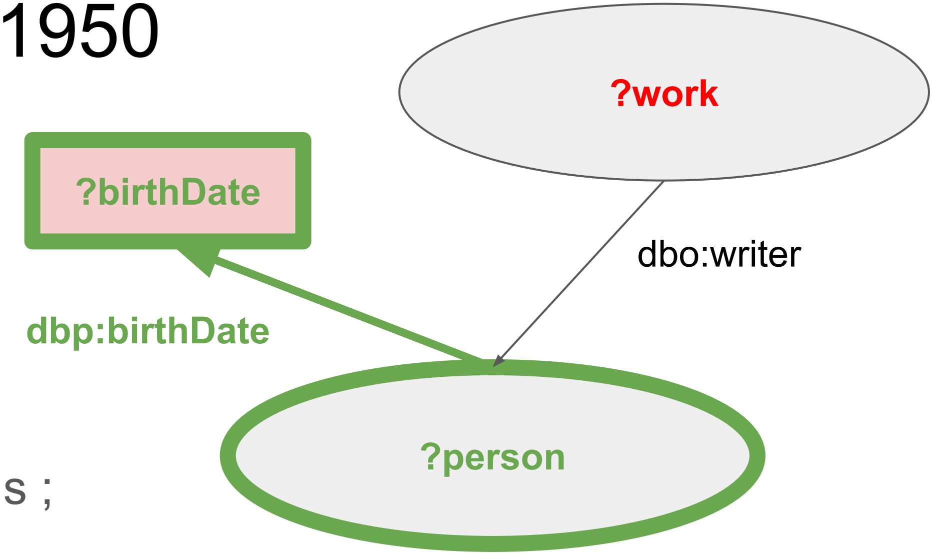
Select all the Italian writers

```
SELECT ?person

WHERE{

        ?person rdf:type dbo:Writer ;

                    dbp:nationality dbr:Italians .

}

LIMIT 10
```



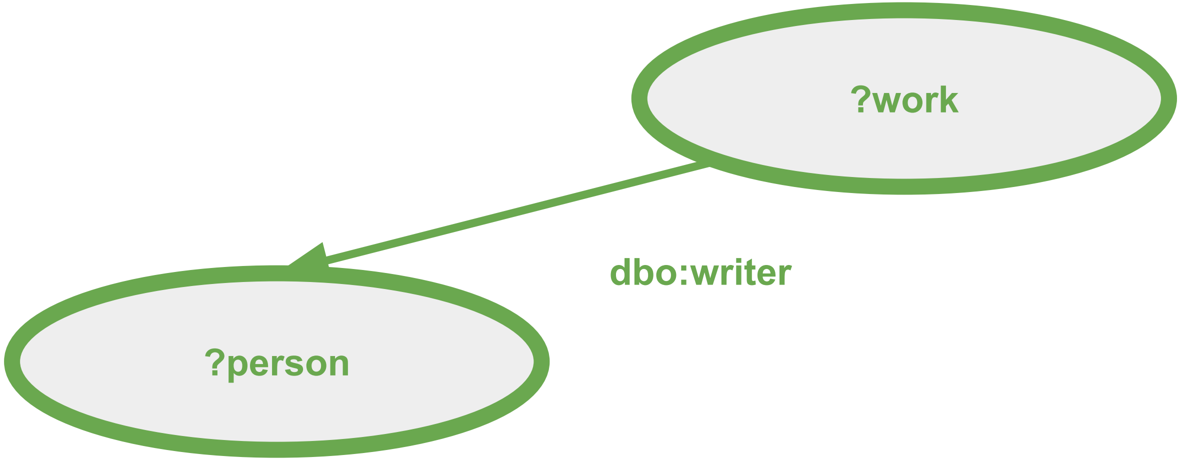
Born between 1900 and 1950

```
SELECT ?person ?birthDate

WHERE{

    ?person rdf:type dbo:Writer ;

            dbp:nationality dbr:Italians ;

            dbp:birthDate ?birthDate .

            FILTER (?birthDate >= "1500"^^xsd:date  &&
                    ?birthDate <= "1900"^^xsd:date)
}
```



Who wrote at least two works

```
SELECT ?person COUNT(?work) AS ?nwork

WHERE{

        ?person rdf:type dbo:Writer ; dbp:birthDate ?birthDate; dbp:nationality dbr:Italians .

        FILTER (?birthDate >= "1500"^^xsd:date  && ?birthDate <= "1900"^^xsd:date)

        ?work dbo:author ?person .

}

GROUP BY ?person

HAVING(COUNT(?work) > 1)
```



Another complex query

Select all Leonardo da Vinci's works, which are hosted by the Louvre

http://dbpedia.org/page/Leonardo_da_Vinci

Another complex query

```
SELECT ?person ?work

WHERE

{

    ?person foaf:name "Leonardo da Vinci"@en .

    ?work    dbo:author ?person;

             dbo:museum dbr:Louvre.

}
```

A simpler solution

```
SELECT ?work

WHERE

{

      ?work    dbo:author <http://dbpedia.org/resource/Leonardo_da_Vinci>;

               dbo:museum dbr:Louvre.

}
```

Another exercise

Select all Italian Presidents and list them in the correct order of mandate

A possible strategy

- Search on DBpedia Sergio Mattarella, the current Italian President
  - http://dbpedia.org/resource/Sergio_Mattarella
- Look at his properties and search for a property which indicates that he is the Italian president
  - dct:subject dbc:Presidents_of_Italy
- Search on DBpedia another Italian President, e.g. Giorgio Napolitano and check if he also contains the property dct:subject dbc:Presidents_of_Italy

YES THERE IS!

- Write the first part of the query

Select all the Italian Presidents

```
SELECT ?president

WHERE {

    ?president dct:subject dbc:Presidents_of_Italy .

}
```

Presidents must be ordered by ascending mandate

- Search for a property that contains a progressive number indicating the number of mandate
  - two properties:
    - dbo:office
    - dbo:orderInOffice
  - select the UNION of the properties
  - filter only the strings containing the word “President of Italy”

Select office

```
SELECT ?president ?office WHERE {
    ?president dct:subject dbc:Presidents_of_Italy .
    {?president dbo:office ?office .
    FILTER (regex(str(?office), "President of Italy" ))}
    UNION
    {?president dbo:orderInOffice ?office .
    FILTER (regex(str(?office), "President of Italy" ))}
}
```

Select the progressive number of the office

- extract a substring
    - if the string starts with a number followed by a character, take only one character and convert it in integer
    - else
        - if the string starts with two characters (i.e. Gronchi), set the number to 3 (Gronchi was the 3rd Italian President)
        - else
            - take two characters and convert them to integer
- order result by asc

IF function form

IF (*expression1*, *expression2*, *expression3*)

The IF function form evaluates the first argument, interprets it as an effective boolean value (EBV), then returns the value of expression2 if the EBV is true, otherwise it returns the value of expression3.

Order by

```
SELECT ?president ?office ?off WHERE {
?president dct:subject dbc:Presidents_of_Italy .
{       ?president dbo:office ?office .
        FILTER (regex(str(?office), "President of Italy" ))}
UNION{
        ?president dbo:orderInOffice ?office .
        FILTER (regex(str(?office), "President of Italy" ))}
bind(
        IF(regex(?office, "^[0-9][a-zA-Z]") = 1, xsd:integer(substr(?office, 1,1)),
        IF(regex(?office, "^[a-zA-Z]+") = 1, 3, xsd:integer(substr(?office,1,2)))) as
?off)
}
ORDER BY ASC(?off)
```

Other functions

- **BOUND** (variable var)
  - Returns true if var is bound to a value. Returns false otherwise. Variables with the value NaN or INF are considered bound.
- **COALESCE**(*expression, ...*)
  - The COALESCE function form returns the RDF term value of the first expression that evaluates without error. In SPARQL, evaluating an unbound variable raises an error.
- **NOT EXISTS** { pattern }
  - Returns false if pattern matches. Returns true otherwise.

Other functions (cont.)

- **EXISTS** { pattern }
  - Returns true if pattern matches. Returns false otherwise.
- **IN** (*expression*, *...*)
  - tests whether the RDF term on the left-hand side is found in the values of list of expressions on the right-hand side. The test is done with "=" operator, which tests for the same value
- **NOT IN** (*expression*, *...*)
  - The NOT IN operator tests whether the RDF term on the left-hand side is not found in the values of list of expressions on the right-hand side.

Conclusions

- To query an RDF dataset we need to know
    - entities URIs
    - deep knowledge of the ontology
    - the SPARQL language