CITY
UNIVERSITY OF LONDON
EST 1894

INM713 Semantic Web Technologies and Knowledge Graphs

Laboratory 3: Querying RDF-based Knowledge Graphs via SPARQL 1.0

Ernesto Jiménez-Ruiz

Academic course: 2020-2021
Updated: February 10, 2021

Contents

1 Git Repositories

Support codes for the laboratory sessions are available in *GitHub*. There are two repositories, one in Python and another in Java:

`https://github.com/city-knowledge-graphs`

2 SPARQL Playground

SPARQL Playground `http://sparql-playground.sib.swiss/` is a framework to learn SPARQL developed by researchers from the Swiss Institute of Bioinformatics `https://www.sib.swiss/`.

The SPARQL Playground has a very intuitive dataset to practice with both simple and sophisticated queries. Figure 1 shows a simplified version of the data and ontology. The same environment has also been used over more complex scenarios to understand the neXtProt and UniProt (knowledge bases about proteins) RDF models.

Task 2.1: Create the following queries. Test them using both the SPARQL Playground interface and programmatically in Python or Java. Use the `playground.ttl` dataset, and the codes in `queryLocalRDFGraph.py` and `QueryLocalRDFGraph.java` from the GitHub repositories (lab3) as examples.

Query 2.1 Query to return Eve's grandfather.

Query 2.2 Things that are dogs with color and sex. (Tip: give a look to the data in `playground.ttl`)

Query 2.3 This query shows pets with their owners (Tip: owner may not exist)

Query 2.4 Select people with their gender and birth date ordered by gender and birth date (oldest first).

3 Nobel Prize Knowledge Graph

The Nobel Prize dataset contains information about Nobel prizes and Nobel Laureates since 1901 (last update on August 2018). Figure 2 shows the schema of the Nobel Prize dataset. The classes and properties in green are reused classes and properties from established vocabularies like *DBPedia* (db and dbo) and *Friend of a Friend* (foaf).[1] The schema is also available from `http://data.nobelprize.org/terms/`.

The Nobel Prize dataset can be accessed via a SPARQL Endpoint (*i.e.*, access point for the dataset via SPARQL): `http://data.nobelprize.org/sparql`. There is also a Web interface to access the Endpoint: `http://data.nobelprize.org/snorql/`. Some examples are also available here: `https://www.nobelprize.org/about/linked-data-examples/`.

[1] `https://www.nobelprize.org/about/linked-data-documentation/`

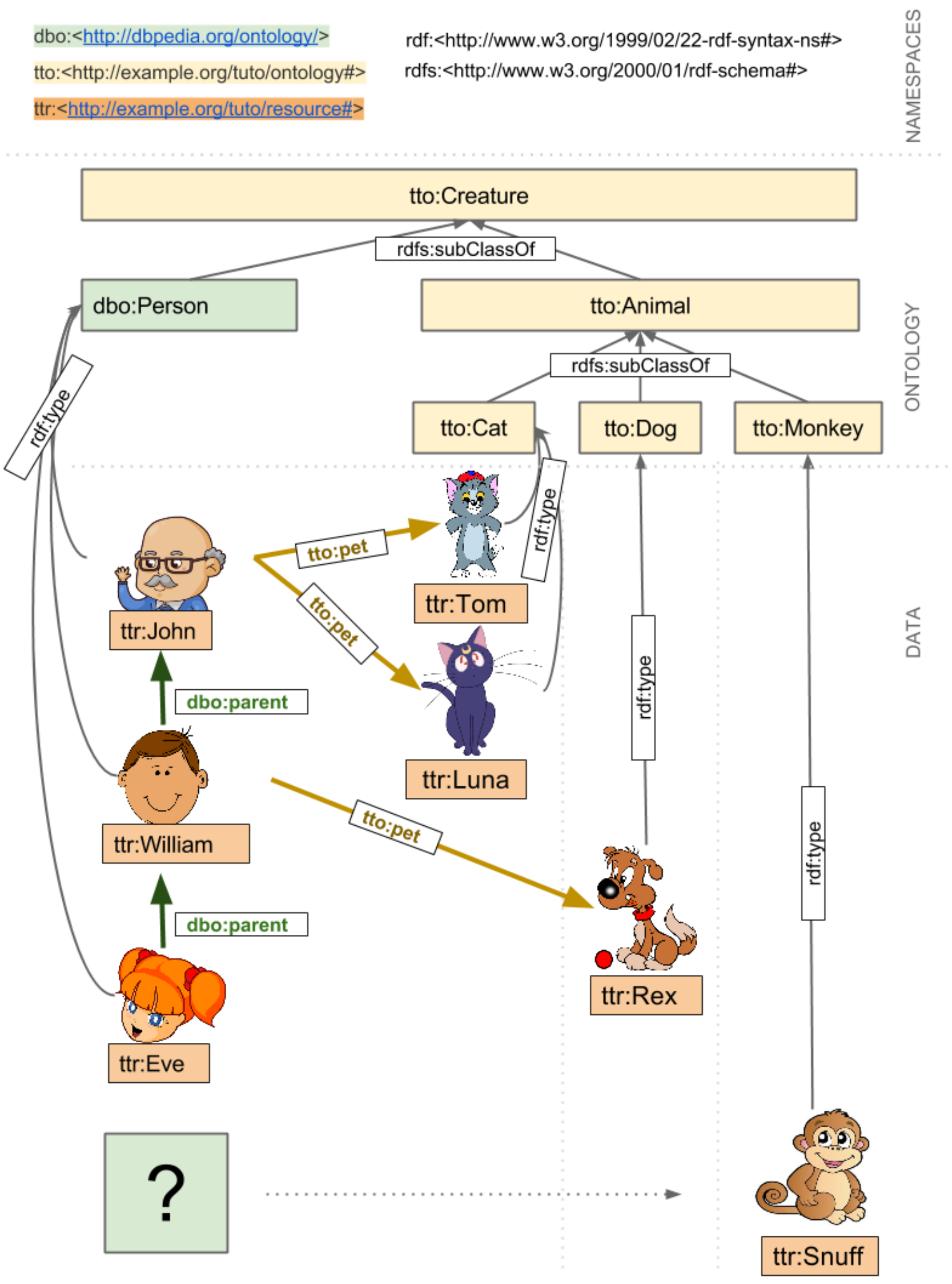

Figure 1: Simplified diagram of the data and "ontology" (from SPARQL Playground).

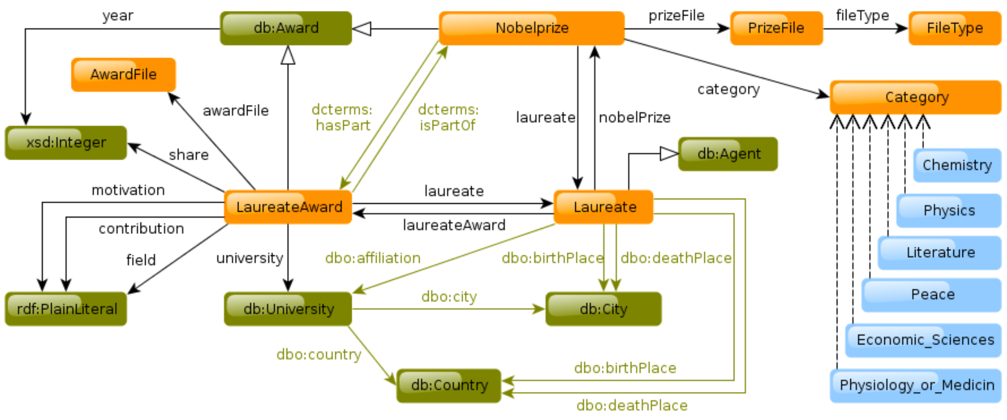

Figure 2: Schema Nobel Prize dataset.

Task 3.1: Create the following queries. Test them programmatically in Python or Java. Use the Nobel Prize SPARQL Endpoint. The codes in `querySPARQLEndpoint.py` and `QuerySPARQLEndpoint.java` from the GitHub repositories (lab3) can be used as reference.

Tip: The Web interface of the Nobel Prize Endpoint may be useful to perform quick tests and explore the data, *e.g.*,:

```
SELECT DISTINCT * WHERE {
<http://data.nobelprize.org/resource/laureate/260> ?p ?o .
}
```

Query 3.1 Find all the Nobel Laureates from the UK.

Query 3.2 Find all the Nobel Laureates who are female and were born after 1949. (Tip: use the function `year()`)

Query 3.3 List all the Nobel Laureates ordering them by discipline for which they were awarded the prize. List the names in alphabetical order.

Query 3.4 Find all the Nobel Laureates born in the US who share the award with someone else. (Tip: use the function `xsd:integer()`)

Query 3.5 List the Laureates born in Italy or Spain.

4 DBPedia Knowledge Graph

During the lecture we have seen several example queries about “Johnny Depp”. These queries are based on the *DBpedia* Knowledge Graph, a RDF version of Wikipedia (`https://dbpedia.org/`), and can be tested via its SPARQL Endpoint (`http://dbpedia.org/sparql`).

Task 4.1: Try the queries in the lecture using the DBPedia Endpoint programmatically or via its Web interface.

5 Solutions

The solutions assume the relevant prefixes have been defined.

Query 2.1:

```
SELECT ?grandfather where {
    ttr:Eve dbo:parent/dbo:parent ?grandfather .
}
```

Query 2.1: (alternative 1)

```
SELECT ?grandfather where {
    ttr:Eve dbo:parent [ dbo:parent ?grandfather ] .
}
```

Query 2.1: (alternative 2)

```
SELECT ?grandfather where {
    ttr:Eve dbo:parent ?x .
    ?x dbo:parent ?grandfather .
}
```

Query 2.2:

```
SELECT ?dogs ?color ?sex WHERE {
    ?dogs rdf:type tto:Dog ;
          tto:sex ?sex;
          tto:color ?color .
}
```

Query 2.3:

```
SELECT ?pet ?owner where {
    ?pet a [ rdfs:subClassOf tto:Animal ].
    OPTIONAL {?owner tto:pet ?pet}
}
```

Query 2.4:

```
SELECT ?people ?gender ?bd WHERE {
    ?people rdf:type dbo:Person ;
            tto:sex ?gender ;
            dbp:birthDate ?bd .
}
ORDER BY ?gender ASC(?bd)
```

Query 3.1:

```
SELECT DISTINCT ?label WHERE {
    ?laur rdf:type nobel:Laureate .
    ?laur rdfs:label ?label .
    ?laur dbpedia-owl:birthPlace nobel-country:United_Kingdom .
}
```

Query 3.2:

```
SELECT DISTINCT ?label ?date WHERE {
    ?laur rdf:type nobel:Laureate ;
          rdfs:label ?label ;
          foaf:gender "female" ;
          foaf:birthday ?date .
    FILTER(year(?date) > 1949)
}
```

Query 3.3:

```
SELECT DISTINCT ?discipline ?label WHERE {
    ?laur rdf:type nobel:Laureate ;
          rdfs:label ?label ;
          nobel:nobelPrize ?n .
    ?n nobel:category ?discipline .
}
ORDER BY ?discipline ASC(?label)
```

Query 3.4:

```
SELECT DISTINCT ?label ?share WHERE {
    ?laur rdf:type nobel:Laureate ;
          rdfs:label ?label ;
          dbpedia-owl:birthPlace nobel-country:USA ;
          nobel:laureateAward ?award .
    ?award nobel:share ?share .
FILTER(xsd:integer(?share)>1)
}
```

Query 3.5:

```
SELECT DISTINCT ?label ?share WHERE {
    ?laur rdf:type nobel:Laureate .
    ?laur rdfs:label ?label .
    {
        ?laur dbpedia-owl:birthPlace nobel-country:Italy .
    }
    UNION{
        ?laur dbpedia-owl:birthPlace nobel-country:Spain .
    }
}
```