IN3067/INM713 Semantic Web Technologies and Knowledge Graphs

Laboratory 4: Querying RDF-based Knowledge Graphs via SPARQL 1.0

Ernesto Jiménez-Ruiz

Academic course: 2023-2024
Updated: February 27, 2024

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`

For Java developers we use maven to deal with dependencies. For Python developers there is always a `requirements.txt` within each lab folder. It is recommended to use environments, but it is not strictly necessary.

2 SPARQL Playground

SPARQL Playground is a dataset to learn SPARQL developed by researchers from the Swiss Institute of Bioinformatics `https://www.sib.swiss/`.

The SPARQL Playground is 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 4.1: Create the following queries. Test them programmatically in Python or Java. Use the `playground.ttl` data, and the codes in the GitHub repositories (lab4) as example.

Query 4.1 Query to return Eve's grandfather.

Query 4.2 Things that are dogs with color and sex. (Tip: give a look to the data in `playground.ttl`)

Query 4.3 Query that shows pets with their owners (Tip: owner may not exist, *i.e.*, it is optional)

Query 4.4 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* (it uses prefixes *db:* and *dbo:*) and *Friend of a Friend* (foaf). The Nobel Prize dataset can be queried via a Web interface: `http://data.nobelprize.org/sparql`.

Documentation:

- Some examples are available here: `https://www.nobelprize.org/about/linked-data-examples/`.

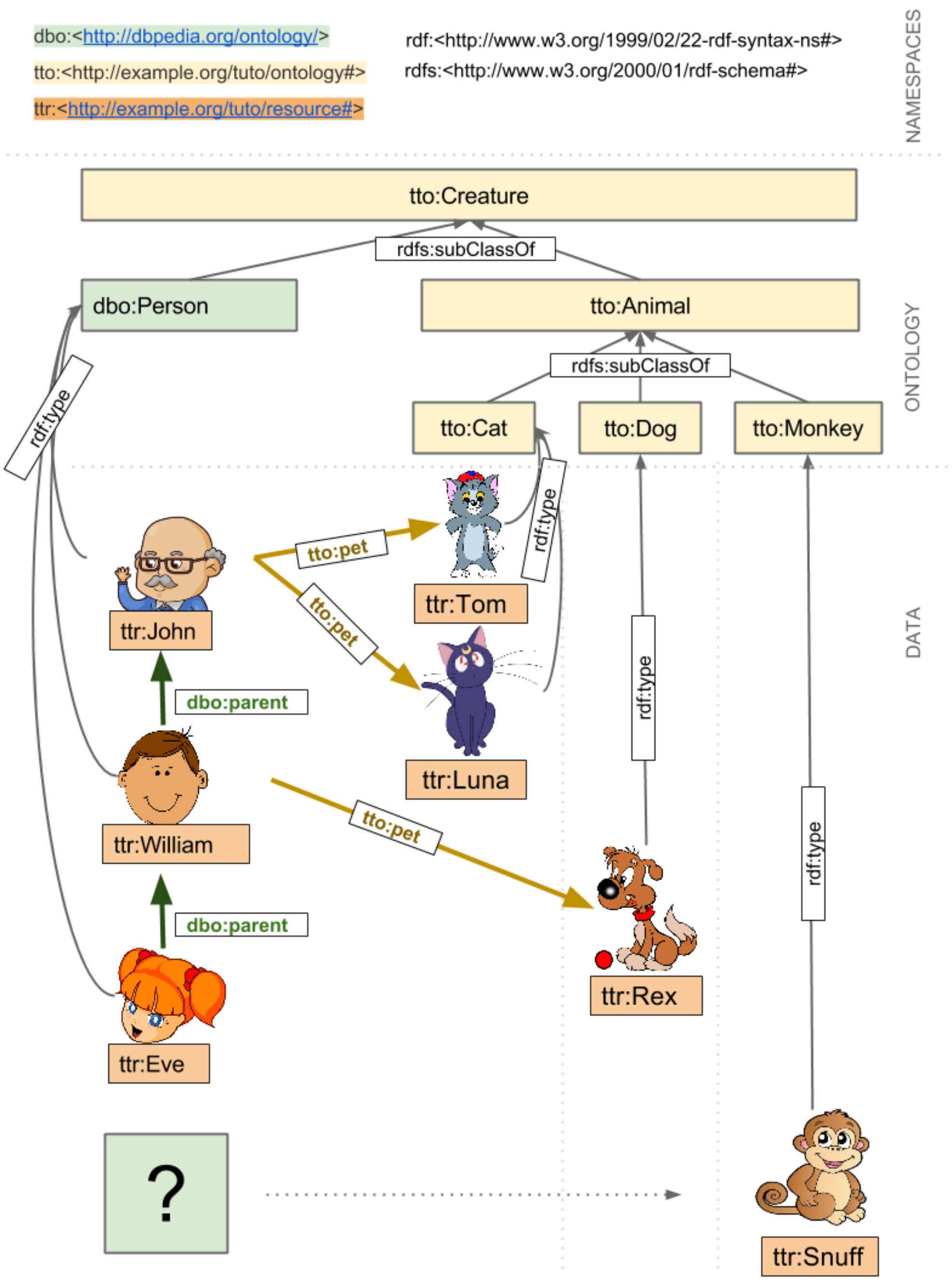

Figure 1: Simplified diagram of the data and "ontology" (from SPARQL Playground).

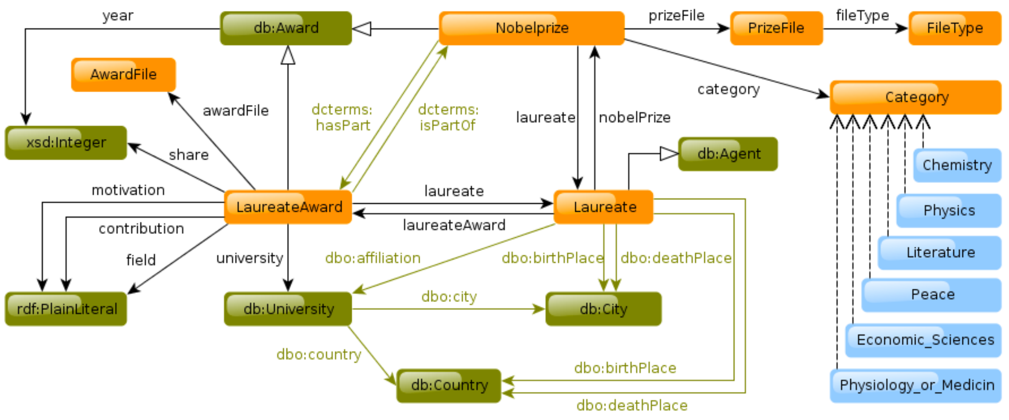

Figure 2: Schema Nobel Prize dataset (from Nobel Prizes as Linked Data).

- Nobel Prizes as Linked Data specification: `https://data.nobelprize.org/specification/`
- The schema/ontology (*nobel-prize-ontology.rdf*) is available in GitHub and also from `http://data.nobelprize.org/terms/`.

Task 4.2: Create the following queries. Test them programmatically in Python or Java. Use the `nobelprize_kg.nt` data. The codes in the GitHub repositories to query a local RDF graph can be used as reference.

Tip: The Web interface 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 4.5 Find all the Nobel Laureates from the UK.

Query 4.6 Find all the Nobel Laureates who are female and were born after 1949. (Tip: use the function `year()`)

Query 4.7 List all the Nobel Laureates ordering them by discipline for which they were awarded the prize. List the names in alphabetical order.

Query 4.8 Find all the Nobel Laureates born in the US who share the award with someone else. (Tip: use the function `xsd:integer()`)

Query 4.9 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:`

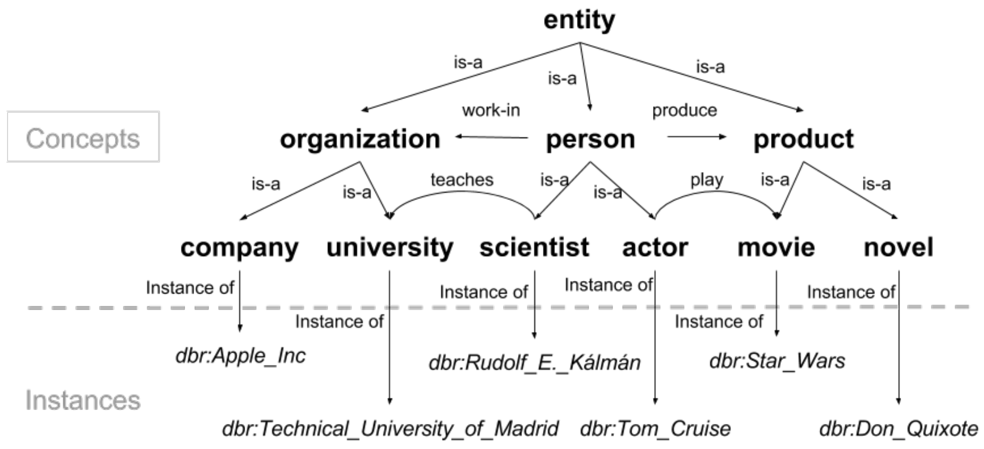

Figure 3: Fragment of DBPedia Knowledge Graph (Figure from `https://gsi-upm.github.io/sematch/`).

`//dbpedia.org/sparql`).[1] The codes in `querySPARQLEndpoint.py` and `QuerySPARQLEndpoint.java` from the GitHub repositories can be used as reference. Figure 3 shows a fragment of the DBPedia KG.

Task 4.3: Try the queries in the lecture using the DBPedia Endpoint programmatically or via its Web interface.

Query 4.10 Modify the running example query from the lecture so that it retrieves performers co-starring with Johnny Depp that were born in Spain. *Tip:* explore `https://dbpedia.org/page/Johnny_Depp`.

5 Solutions

The solutions assume the relevant prefixes have been defined in the 'prologue'. Sometimes, when running queries against an endpoint some of the prefixes are already defined on the endpoint side. In case of doubt (or error), however, it is better if the queries are self-contained with the declaration of the used prefixes. Common general prefixes:

```
PREFIX xsd:  <http://www.w3.org/2001/XMLSchema#>
PREFIX rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:  <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl:  <http://www.w3.org/2002/07/owl#>
```

5.1 SPARQL Playground

Specific prefixes for the SPARQL playground queries.

```
PREFIX tto:  <http://example.org/tuto/ontology#>
PREFIX ttr:  <http://example.org/tuto/resource#>
```

[1]Access point for the dataset via SPARQL.

```
PREFIX dbo:  <http://dbpedia.org/ontology/>
PREFIX dbp:  <http://dbpedia.org/property/>
PREFIX dbpedia:  <http://dbpedia.org/resource/>
```

Query 4.1: using paths

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

Query 4.1: (alternative 1 using blank node syntax)

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

Query 4.1: (alternative 2 without blank node syntax)

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

Query 4.2:

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

Query 4.3:

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

Query 4.3: (alternative without blank node syntax)

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

Query 4.3: (alternative if reasoning is enabled - week 7)

```
SELECT ?pet ?owner where {
    ?pet a tto:Animal .
    OPTIONAL {?owner tto:pet ?pet}
}
```

Query 4.4:

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

5.2 Nobel Prize KG

Specific prefixes for the Nobel Prize KG queries.

```
PREFIX nobel:  <http://data.nobelprize.org/terms/>
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
PREFIX dbpedia-owl:  <http://dbpedia.org/ontology/>
PREFIX nobel-country:  <http://data.nobelprize.org/resource/country/>
```

Query 4.5:

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

Query 4.6:

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

Query 4.7:

```
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 4.8:

```
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 4.9:

```
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 .
    }
}
```

5.3 DBpedia KG

Query 4.10:

```
SELECT DISTINCT ?costar WHERE {
    ?jd foaf:name "Johnny Depp"@en .
    ?m dbo:starring ?jd .
    ?m dbo:starring ?other .
    ?other foaf:name ?costar .
    ?other dbo:birthPlace ?place .
    ?place dbo:country dbr:Spain
    FILTER (STR(?costar)!="Johnny Depp")
}
ORDER BY ?costar
```