SPARQL

Read

- Foundations of Semantic Web Technologies: chapter 7.

1 Query engine

In this exercise you are asked to make a SPARQL query engine.

1.1 Exercise

Write a java program which reads an RDF graph and a SPARQL query from file, queries the graph and outputs the query results as a table. Your program should accept `SELECT` queries, `CONSTRUCT` queries and `ASK` queries. A messages should be given if the query is of a different type.

1.1.1 Tip

If I query the Simpsons RDF graph (`simpsons.ttl`) we wrote in a previous exercise with my SPARQL query engine and the `SELECT` query

```
1  PREFIX sim: <http://www.ifi.uio.no/INF3580/simpsons#>
2  PREFIX fam: <http://www.ifi.uio.no/INF3580/family#>
3  PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
4  PREFIX foaf: <http://xmlns.com/foaf/0.1/>
5  SELECT ?s ?o
6  WHERE{ ?s foaf:age ?o }
7  LIMIT 1
```

I get[1] the following: (To get the nicely formatted output I use the class ResultSetFormatter.)

```
-------------------------------------------------------------------
| s                                                 | o           |
===================================================================
| <http://www.ifi.uio.no/INF3580/simpsons#Maggie> | "1"^^xsd:int |
-------------------------------------------------------------------
```

Executing with the `ASK` query

```
1  ASK{ ?s ?p ?o }
```

gives me

```
true
```

[1] Note that your results may be different according to how your Simpsons RDF file looks like.

Executing with the `CONSTRUCT` query

```
1  PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
2  PREFIX fam: <http://www.ifi.uio.no/INF3580/family#>
3  PREFIX sim: <http://www.ifi.uio.no/INF3580/simpsons#>
4  PREFIX foaf: <http://xmlns.com/foaf/0.1/>
5  CONSTRUCT{ sim:Bart rdfs:label ?name }
6  WHERE{ sim:Bart foaf:name ?name }
```

gives me

```
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix foaf:    <http://xmlns.com/foaf/0.1/> .
@prefix xsd:     <http://www.w3.org/2001/XMLSchema#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix sim:     <http://www.ifi.uio.no/INF3580/simpsons#> .
@prefix fam:     <http://www.ifi.uio.no/INF3580/family#> .
sim:Bart rdfs:label "Bart Simpsons" .
```

2 DBpedia

DBpedia is, according to DBpedia,

> a community effort to extract structured information from Wikipedia and to make this information available on the Web. DBpedia allows you to ask sophisticated queries against Wikipedia, and to link other data sets on the Web to Wikipedia data.

We will use their SPARQL endpoint, http://dbpedia.org/sparql, to extract some information.

There is also a more fancy GUI available, if you can get anything out of it: http://dbpedia.org/isparql/.

Links:

- http://virtuoso.openlinksw.com/dataspace/dav/wiki/Main/VOSSparqlProtocol

Note that DBpedia can be slow to respond.

2.1 Exercise

Extend your query engine so that it can query a SPARQL endpoint and not just a file.

A simple solution to differentiate between if the source to query is a file or a SPARQL endpoint is to let the program read three arguments, where the first argument specifies the source, e.g., running

```
java_program file simpsons.ttl sparql_query.rq
```

results in running the query `sparql_query.rq` on the file `simpsons.ttl`, just like the query engine you have already written in a previous exercise, while running

```
java_program endpoint http://some.sparql/endpoint/ sparql_query.rq
```

returns the result of querying the endpoint `http://some.sparql/endpoint/` with the query `sparql_query.rq`.

2.2 Exercise

Find all Simpsons characters in DBpedia and list their name, gender and relatives. Let gender and relatives be optional and list only names in English. Assume all characters have a `dcterms:subject` relation to Category:The_Simpsons_characters[2] , i.e.,

```
?character dcterms:subject
    <http://dbpedia.org/resource/Category:The_Simpsons_characters> .
```

Browse DBpedia to find the correct resource and property identifiers to use in your query.

2.3 Exercise

Make a `CONSTRUCT` query which creates a graph based on the `SELECT` query you made above. Type a character as `foaf:Person` and use the properties

- `foaf:name`
- `dbpfam:hasGenderResource`
- `dbpfam:hasGenderLiteral`
- `dbpfam:hasRelative`
- `rdfs:label`

to relate the person to its name, gender and relatives. Use `hasGenderLiteral` if the value of gender is a literal, and `hasGenderResource` if the value is a resource[3]. The object values for `fam:hasRelationshipTo` must be resources and not literals. This means that you should ignore values of relatives given as literals. `rdfs:label` shall hold the name of the character.

2.3.1 Tip

Assume the table

Person	Name	Gender	Relative
`dbp:Marge_Simpson`	"Marge Simpson"	"Female"	`dbp:Maggie_Simpson`
`dbp:Lisa_Simpson`	"Lisa Simpson"	`dbp:Female`	"Father: Homer"

is the result of running your `SELECT` query. Then your `CONSTRUCT` query should produce the following RDF graph

```
dbp:Marge_Simpson a foaf:Person ;
  foaf:name "Marge Simpson" ;
  fam:hasGenderLiteral "Female" ;
  fam:hasRelative dbp:Maggie_Simpson ;
  rdfs:label "Marge Simpson" .

dbp:Lisa_Simpson a foaf:Person ;
 foaf:name "Lisa Simpson" ;
 fam:hasGenderResource dbp:Female ;
 rdfs:label "Lisa Simpson" .
```

[2] http://dbpedia.org/resource/Category:The_Simpsons_characters

[3] We create this odd construction because we are going to re-use this ontology later, and in OWL DL object properties and datatype properties are disjoint.

2.4 Exercise

Explain what a `DESCRIBE` SPARQL query is. Make an example using the DBpedia SPARQL endpoint.