CSI 5180 Topics in AI - Ontologies and Semantic Web
Winter 2019

Assignment 1 - RDF and Sparql

Objective	Become familiar with the RDF data model. Learn about the Semantic Web framework Jena. Learn about Sparql. Explore DBpedia as a SW knowledge base.
Due date	January 31rst, 11:30pm.
What to submit ?	A report, in pdf format, which will include a title page (name, student number) as well as screenshots, bits of code, bits of RDF, etc, as asked for each of the six questions below.
How to submit ?	In Brightspace, through the link provided under the Assignment module.
Penalty	-10% per day late
Percentage	15% of overall semester grade
Software Requirement	This assignment will make use of Apache Jena (http://jena.apache.org/). You are also encouraged to explore RDF editors, and visualization tools.

1. Creating an small RDF model of *YOU*

This first step is to create a small data model that will represent you and a few of your friends or family members. The purpose is to manually build such small model to get familiar with various vocabularies (VCARD, FOAF, etc) and also with RDF modeling.

You should make use of the following vocabularies:

Vocabulary	prefix
RDF https://www.w3.org/1999/02/22-rdf-syntax-ns	rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
vCard vocabulary for describing people and organizations https://www.w3.org/TR/vcard-rdf/	vCard: <http://www.w3.org/2001/vcard-rdf/3.0#>
foaf http://xmlns.com/foaf/spec/	foaf: <http://xmlns.com/foaf/0.1/>
DBpedia https://wiki.dbpedia.org	dbr: <http://dbpedia.org/resource/>
local Made up for your example	ex: <http://csi5180-example.org/>

Your data model should include:

- A resource corresponding to yourself.
- A set of properties for yourself:
    - fullname, family name, given name, email (this can all use the vCard properties and classes)
    - that you are registered in courses (put at least 2) and that these courses are offered at University of Ottawa (or Carleton); make sure to use the DBpedia entry for the university
    - where you live (let's pretend you live in Ottawa for now….), include the DBpedia entry for Ottawa
- Add two other resources for two friends or family member who live in a different city than Ottawa. For each of them, provide the same properties as above. Make sure the university or organization you provide for them exists in DBpedia. *(you can just make it up!)*

Do not explicitly include the city of the university or organization where you or your friends live. We will infer it later through properties within DBpedia.

The simples "human-friendly" serialization of RDF is TURTLE. Here is a small partial example for me that you can look at to get you started.

```
@prefix rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix vCard: <http://www.w3.org/2001/vcard-rdf/3.0#> .
@prefix foaf:  <http://xmlns.com/foaf/0.1/> .
@prefix dbr:    <http://dbpedia.org/resource/> .
@prefix ex:    <http://csi5180-example.org/> .

ex:CB
      vCard:FN  "Caroline Barriere" ;
      vCard:N   [ vCard:Family  "Barriere" ;
                  vCard:Given   "Caroline"
            ] ;
      vCard:locality        dbr:Ottawa ;
      ex:teaches   ex:CSI5180 ;
      foaf:knows   ex:DR  .

ex:CSI5180  ex:offeredAt dbr:University_of_Ottawa .

ex:DR
      vCard:FN  "Denise Richard" ;
      vCard:N   [ vCard:Family  "Richard" ;
                  vCard:Given   "Denise"
            ] ;
      vCard:locality        dbr:Toronto  .
```

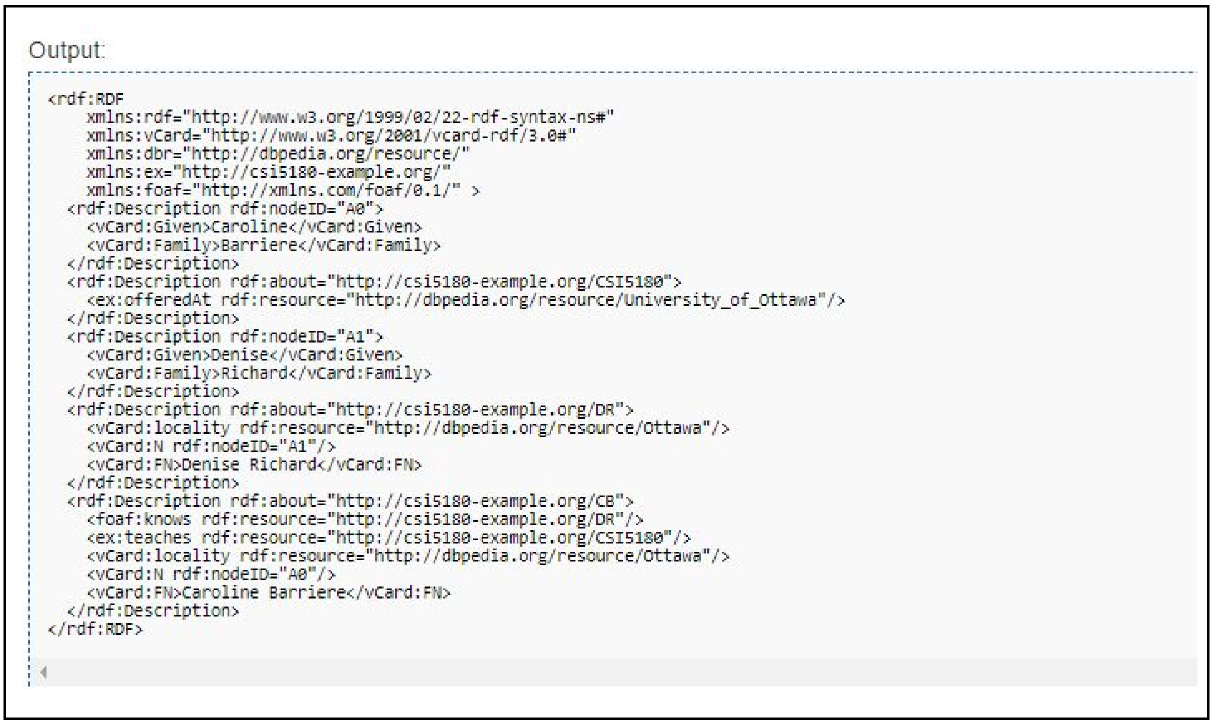
Figure 1 - Possible content of file CB.ttl (turtle file)

TO INCLUDE IN REPORT:

For this question, you should include the content of your turtle file (as shown above).

2. Testing validators and converters

We have seen in class that there are different serializations for the RDF model.
Try using a converter, such as http://rdfvalidator.mybluemix.net/ to convert your turtle file into an RDF/XML file. Although less readable for humans, the RDF/XML format is often the one required by applications or programs.



Output:

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:vCard="http://www.w3.org/2001/vcard-rdf/3.0#"
    xmlns:dbr="http://dbpedia.org/resource/"
    xmlns:ex="http://csi5180-example.org/"
    xmlns:foaf="http://xmlns.com/foaf/0.1/" >
  <rdf:Description rdf:nodeID="A0">
    <vCard:Given>Caroline</vCard:Given>
    <vCard:Family>Barriere</vCard:Family>
  </rdf:Description>
  <rdf:Description rdf:about="http://csi5180-example.org/CSI5180">
    <ex:offeredAt rdf:resource="http://dbpedia.org/resource/University_of_Ottawa"/>
  </rdf:Description>
  <rdf:Description rdf:nodeID="A1">
    <vCard:Given>Denise</vCard:Given>
    <vCard:Family>Richard</vCard:Family>
  </rdf:Description>
  <rdf:Description rdf:about="http://csi5180-example.org/DR">
    <vCard:locality rdf:resource="http://dbpedia.org/resource/Ottawa"/>
    <vCard:N rdf:nodeID="A1"/>
    <vCard:FN>Denise Richard</vCard:FN>
  </rdf:Description>
  <rdf:Description rdf:about="http://csi5180-example.org/CB">
    <foaf:knows rdf:resource="http://csi5180-example.org/DR"/>
    <ex:teaches rdf:resource="http://csi5180-example.org/CSI5180"/>
    <vCard:locality rdf:resource="http://dbpedia.org/resource/Ottawa"/>
    <vCard:N rdf:nodeID="A0"/>
    <vCard:FN>Caroline Barriere</vCard:FN>
  </rdf:Description>
</rdf:RDF>
```

Figure 2 - CB.ttl (turtle file) converted to RDF/XML format

<u>TO INCLUDE IN REPORT:</u>

For this question, you should include the output of the converter, as above.

3. Visualization

It is nice to see the graphical representation of the data model. There are different ways to transform the RDF into a graph to visualize it. I let you explore, and decide how you wish to do this. One simple way is to use an online visualization program, such as http://visgraph3.org/. This is really not the most flexible option, but it works. If you have time, a more flexible approach would be to use a graph package in Java, and then transform your RDF into a graph.

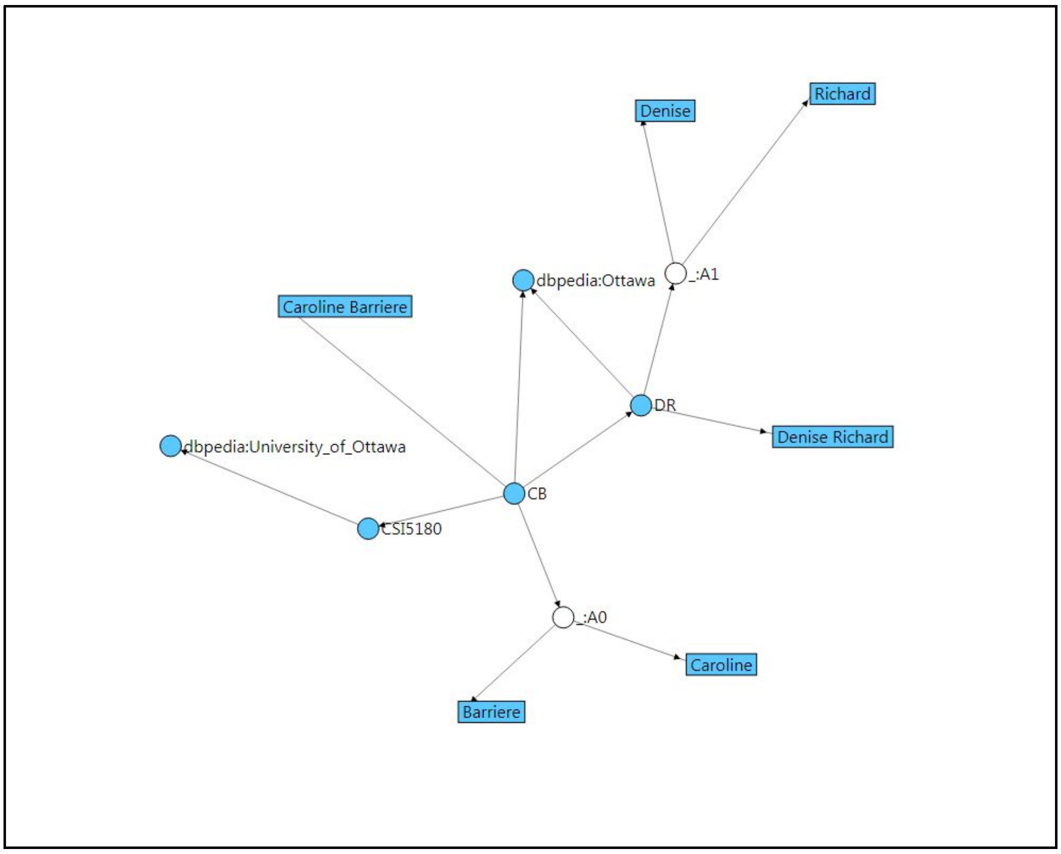


Figure 3 - CB.rdf uploaded to the visualizer

<u>TO INCLUDE IN REPORT:</u>

For this question, you should include the output of your visualizer, as above. Please mention which visualizer you used, or how you transformed your RDF into a graph to be shown.

4. Querying your model within Jena

We will now start using Jena (http://jena.apache.org/). You will need to download Jena and include it in your Java Project.

You can take a look at the jena rdf tutorial: http://jena.apache.org/tutorials/rdf_api.html
This will help you get started.

We will use Jena to load the data model we have previously built and question it, using Sparql queries. For now, the queries will be simple, and only about the data included in the model.

Queries:
1. List courses that you take at the University.
2. List the given names of people you know.
3. List the organizations (schools) where your friends (people you know) work or study.
4. (your choice - make up a query)
5. (your choice - make up a query)

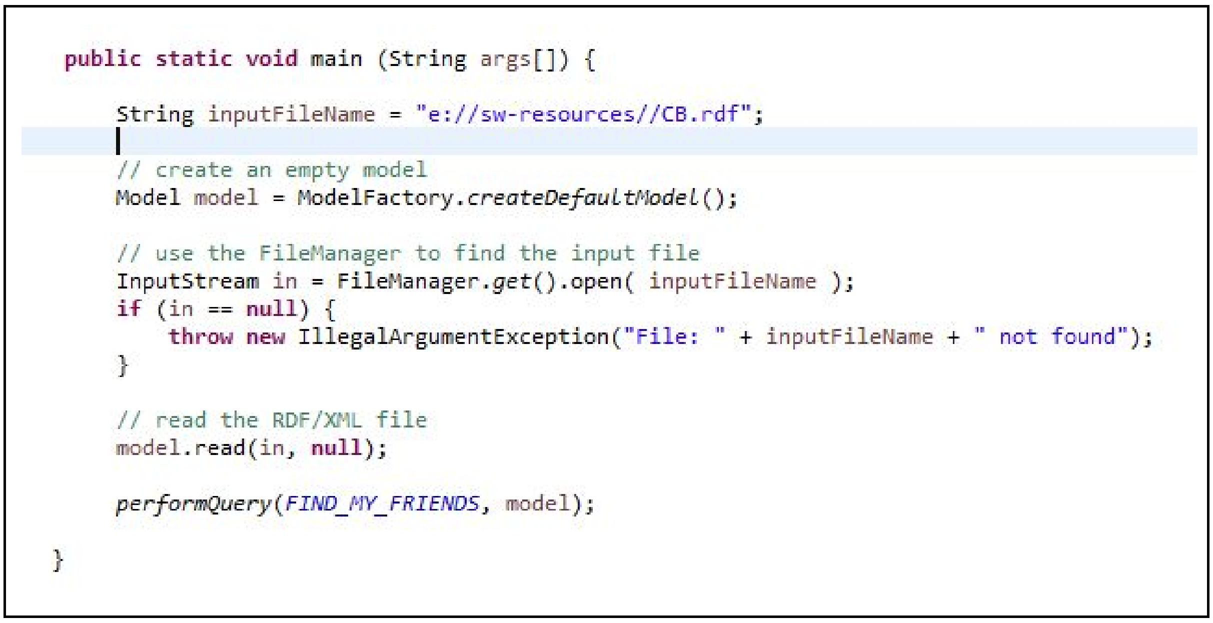
Here is a bit of the java code to help you get started. The code assumes that jena has been downloaded and is available to your project (in Eclipse or other development environment).

In Figure 4, we see that the file CB.rdf (as generated earlier in Question 2) is read as a model. And then the model is queried.

Then in Figure 5, the prefixes are provided as constants. A first query string, to answer a query (list the full name of people you know) is written as a Sparql query.

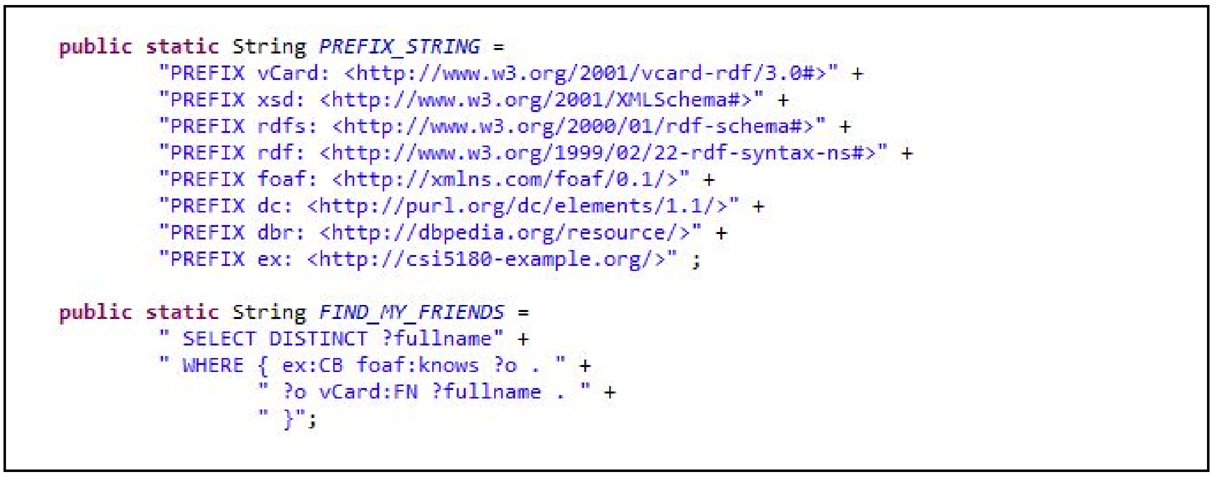
Finally in Figure 6, the method performQuery is shown.

Please modify and adapt as you wish to make the code your own. To learn more about Sparql queries, there are tutorials online. There is one within Jena http://jena.apache.org/tutorials/sparql.html, but you can find others as well.



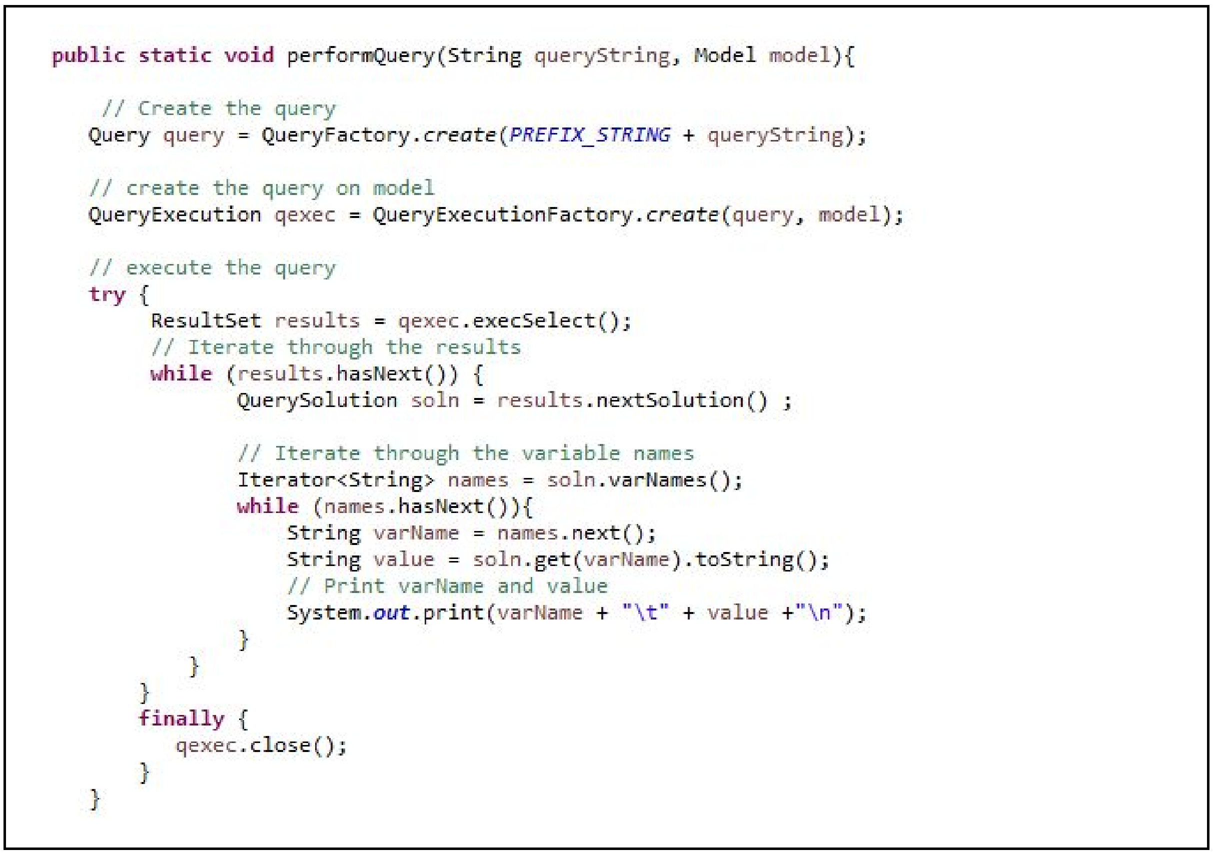
```
public static void main (String args[]) {

    String inputFileName = "e://sw-resources//CB.rdf";

    // create an empty model
    Model model = ModelFactory.createDefaultModel();

    // use the FileManager to find the input file
    InputStream in = FileManager.get().open( inputFileName );
    if (in == null) {
        throw new IllegalArgumentException("File: " + inputFileName + " not found");
    }

    // read the RDF/XML file
    model.read(in, null);

    performQuery(FIND_MY_FRIENDS, model);

}
```

Figure 4 - The reading of the model (as defined earlier) and calling the performQuery function.



```
public static String PREFIX_STRING =
        "PREFIX vCard: <http://www.w3.org/2001/vcard-rdf/3.0#>" +
        "PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>" +
        "PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>" +
        "PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>" +
        "PREFIX foaf: <http://xmlns.com/foaf/0.1/>" +
        "PREFIX dc: <http://purl.org/dc/elements/1.1/>" +
        "PREFIX dbr: <http://dbpedia.org/resource/>" +
        "PREFIX ex: <http://csi5180-example.org/>" ;

public static String FIND_MY_FRIENDS =
        " SELECT DISTINCT ?fullname" +
        " WHERE { ex:CB foaf:knows ?o . " +
                " ?o vCard:FN ?fullname . " +
                " }";
```

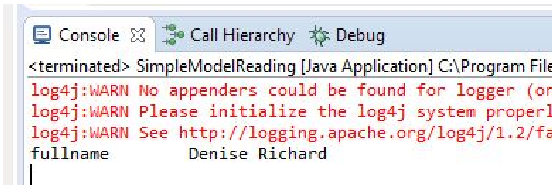
Figure 5 - Prefixes and a single query defined.



```
public static void performQuery(String queryString, Model model){

    // Create the query
   Query query = QueryFactory.create(PREFIX_STRING + queryString);

   // create the query on model
   QueryExecution qexec = QueryExecutionFactory.create(query, model);

   // execute the query
   try {
        ResultSet results = qexec.execSelect();
        // Iterate through the results
        while (results.hasNext()) {
               QuerySolution soln = results.nextSolution() ;

               // Iterate through the variable names
               Iterator<String> names = soln.varNames();
               while (names.hasNext()){
                   String varName = names.next();
                   String value = soln.get(varName).toString();
                   // Print varName and value
                   System.out.print(varName + "\t" + value +"\n");
               }
           }
       }
       finally {
          qexec.close();
       }
   }
```

Figure 6 - Method to perform a query on a model

Figure 7 shows the result of this query. As there is a single person I know "Denise Richard", this is the only fullname obtained.



```
<terminated> SimpleModelReading [Java Application] C:\Program File
log4j:WARN No appenders could be found for logger (or
log4j:WARN Please initialize the log4j system properl
log4j:WARN See http://logging.apache.org/log4j/1.2/fa
fullname        Denise Richard
```

Figure 7 - Result of FIND_MY_FRIENDS query

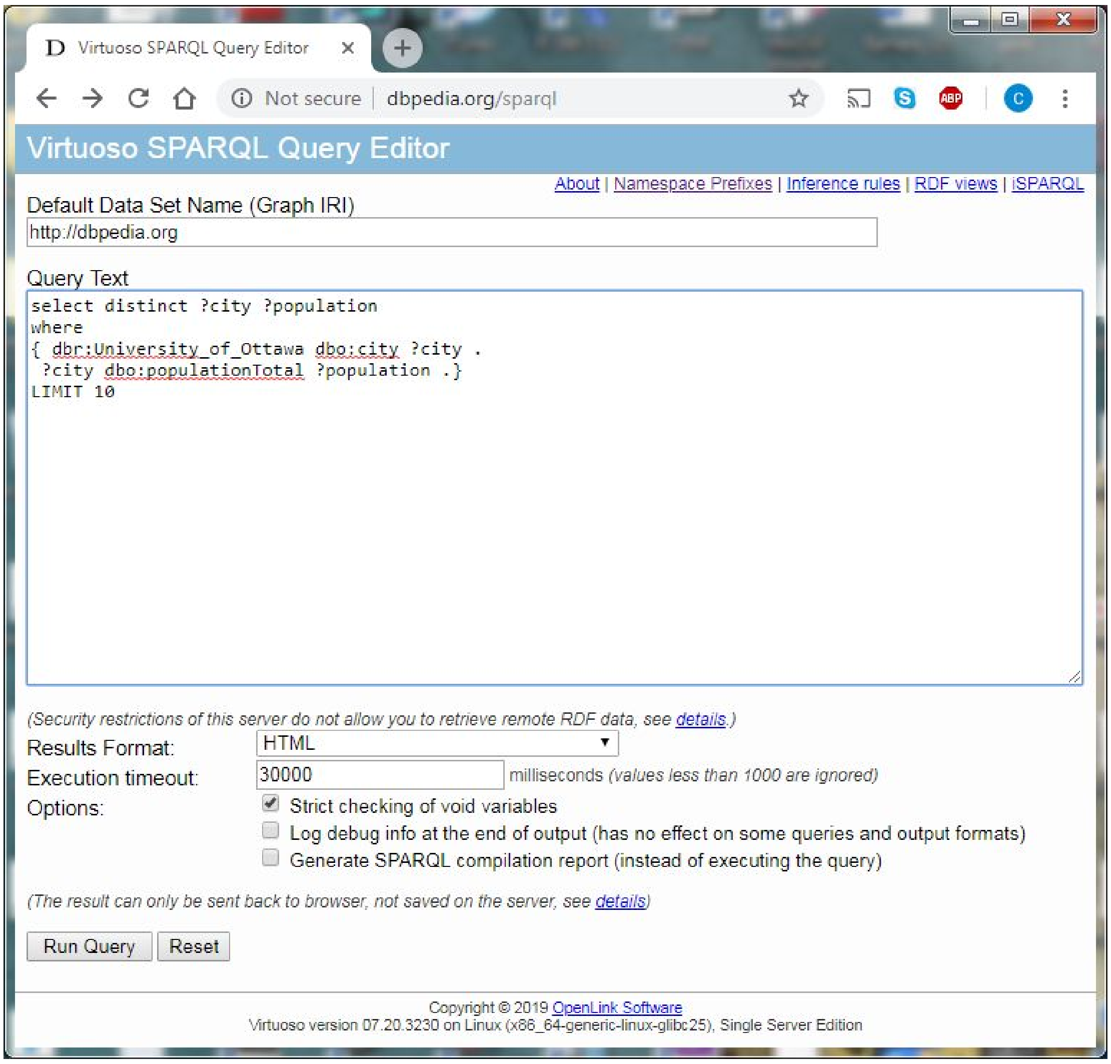
TO INCLUDE IN REPORT:

Show the Sparql queries that you wrote (as in Figure 5) as well as the obtained results (as in Figure 7).

5. Explore DBpedia Sparql endpoint

We will now explore a Sparql endpoint and query it. Sparql endpoints are a bit unstable… This site http://sparqles.ai.wu.ac.at/availability provides the availability. DBpedia is perhaps one of the most stable endpoints, which is nice since the only other option is to download the data and query it locally. As DBpedia is quite large, it is nice to be able to query it at a distance.

You can go to http://dbpedia.org/sparql to access the SPARQL endpoint. In Figure 8, you can see a query about the population of the city in which the University of Ottawa is. And in Figure 9, you see the result when performing "Run Query". The endpoint already knows about many prefixes (see the tab Namespace Prefixes), so we do not need to provide them.


Virtuoso SPARQL Query Editor
Not secure
dbpedia.org/sparql
About | Namespace Prefixes | Inference rules | RDF views | iSPARQL

Virtuoso SPARQL Query Editor

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

Query Text

```
select distinct ?city ?population
where
{ dbr:University_of_Ottawa dbo:city ?city .
 ?city dbo:populationTotal ?population .}
LIMIT 10
```

(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)

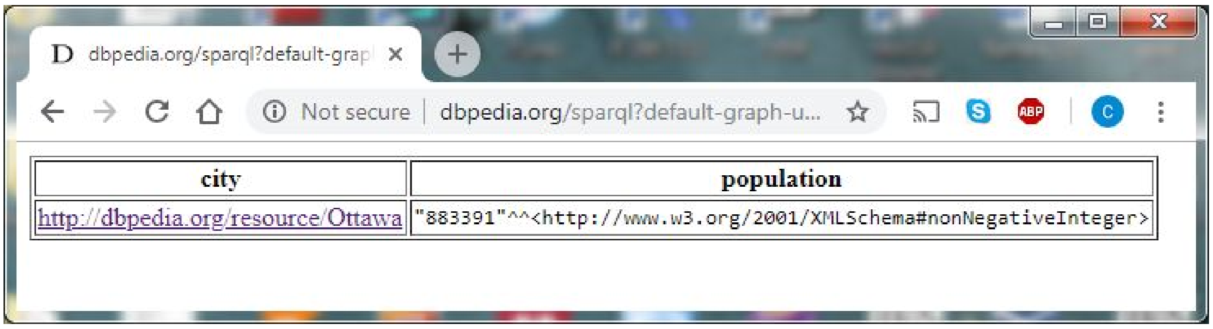
Options:
[✓] Strict checking of void variables
[ ] Log debug info at the end of output (has no effect on some queries and output formats)
[ ] Generate SPARQL compilation report (instead of executing the query)

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

Run Query | Reset

Copyright © 2019 OpenLink Software
Virtuoso version 07.20.3230 on Linux (x86_64-generic-linux-glibc25), Single Server Edition

Figure 8 - Example of a query at the DBpedia SPARQL endpoint


dbpedia.org/sparql?default-grap
Not secure
dbpedia.org/sparql?default-graph-u...

city	population
http://dbpedia.org/resource/Ottawa	"883391"^^<http://www.w3.org/2001/XMLSchema#nonNegativeInteger>

Figure 9 - Result from query in Figure 8

Queries to perform:
1. Show all the universities in Ottawa.
2. Show cities in Canada more populated than Ottawa.
3. Show musicians born (use dbo:MusicalArtist) in Ottawa who are older than 40 years old.
4. (your choice - make up a query after exploring DBpedia pages)
5. (your choice - make up a query after exploring DBpedia pages)

To help you figure out the predicates, you should browse through the different entities. You can start here http://dbpedia.org/page/University_of_Ottawa and navigate.

If you need further SPARQL tutorial: https://www.w3.org/2009/Talks/0615-qbe/ . It provides examples, even if many of the endpoints used are no longer available.

TO INCLUDE IN REPORT:

Show the Sparql queries that you wrote (as in Figure 8) as well as the obtained results (as in Figure 9).

6. Mix local and endpoint queries within Jena

You will now have to investigate how to use an endpoint within Jena. The following queries to be performed will require that you query both your local model, and the DBpedia endpoint to provide answers.

**I'm purposely not helping on this one…. you're on your own! You can do it.*

Queries to perform:
1. Show the populations of the cities where your friends live.
2. Show the list of cities where you friends live which are smaller (in area) than Ottawa.
3. Show the friends who work in the same city where they live.
4. (your choice - make up a query that require both your model and DBpedia)
4. (your choice - make up a query that require both your model and DBpedia)

TO INCLUDE IN REPORT:

Show the SPARQL queries that you wrote, and the results obtained. Also, show the java code in which you combine the part of the query from the model, and the part from DBpedia.