Lab Session #4

Introduction

Welcome to lab #4. This week's lab is about learning the *SPARQL query language* for knowledge graphs. As always, if you did not finish any of the tasks from the previous week(s), make sure you catch up with any remaining tasks, since some of the new tasks build on previous work.

Follow-up Lab #3

Here's an example solution for the FOCU university schema & data from the previous lab. Of course, your solution might look slightly different. If you're confused about any part of it, just ask in the Moodle Discussion forum!

Task #1: SPARQL 101

Here is a nice SPARQL cheat sheet (well, more like cheat slides) that are helpful for getting a quick overview (note that we did not discuss all the details of SPARQL in class). To learn SPARQL it's best to experiment with a number of different queries. Many (but not all) public knowledge graphs provide a public SPARQL interface (a so-called *SPARQL endpoint*), for example this one at DBpedia.

Note that these public, open endpoints are typically very restrictive in terms of query result size, execution time, and query memory use, so don't be surprised if you get an error message instead of a result. Sometimes it helps to retry a query or limiting the result size (e.g., with a LIMIT 50).

Task #1.1: DBpedia

To gain an impression on how powerful graph queries can be, here are two examples that you can try out using DBpedia's public SPARQL endpoint:

Musicians who were born in Berlin:

```
SELECT ?name ?birth ?description ?person
WHERE {
      ?person a dbo:MusicalArtist .
      ?person dbo:birthPlace :Berlin .
      ?person dbo:birthDate ?birth .
      ?person foaf:name ?name .
      ?person rdfs:comment ?description .
```

```
      FILTER (LANG(?description) = 'en') .
} ORDER BY ?name
```

1.

Soccer players, who are born in a country with more than 10 million inhabitants, who played as goalkeeper for a club that has a stadium with more than 30.000 seats and where the club's country is different from the player's birth country:

```
SELECT DISTINCT ?soccerplayer ?countryOfBirth ?team ?countryOfTeam
?stadiumcapacity
WHERE {
     ?soccerplayer a dbo:SoccerPlayer ;
           dbo:position|dbp:position
<http://dbpedia.org/resource/Goalkeeper_(association_football)>;
           dbo:birthPlace/dbo:country* ?countryOfBirth ;
           dbo:team ?team .
     ?team dbo:capacity ?stadiumcapacity ;
           dbo:ground ?countryOfTeam .
     ?countryOfBirth a dbo:Country ;
           dbo:populationTotal ?population .
     ?countryOfTeam a dbo:Country .
     FILTER (?countryOfTeam != ?countryOfBirth)
     FILTER (?stadiumcapacity > 30000)
     FILTER (?population > 10000000)
} ORDER BY ?soccerplayer
```

2.

You can see how intelligent assistants like Watson, Siri, Alexa etc. are able to answer so many questions, by querying their knowledge graphs.

Now, try to write your own queries to determine:

1. All *universities located in Canada*, with their city and *optionally* (if it exists) their home page.
2. All people who *studied at Concordia University* (and are listed in DBpedia), together with their description (in English or any other language you prefer). Hint: look for the *Alma Mater* predicate and make sure you understand its domain & range.

Task #2: Your own SPARQL Server

As discussed in the lecture, there are a number of options for setting up your own SPARQL *endpoint*; here, we will use the open source Apache Fuseki server.

Task #2.1: Getting started with Fuseki

1. Start by downloading the fuseki binary distribution from https://jena.apache.org/download/index.cgi

Unpack the archive and make the server script executable:

```
> tar xf apache-jena-fuseki-3.17.0.tar.gz
> cd apache-jena-fuseki-3.17.0
> chmod u+x fuseki-server
```

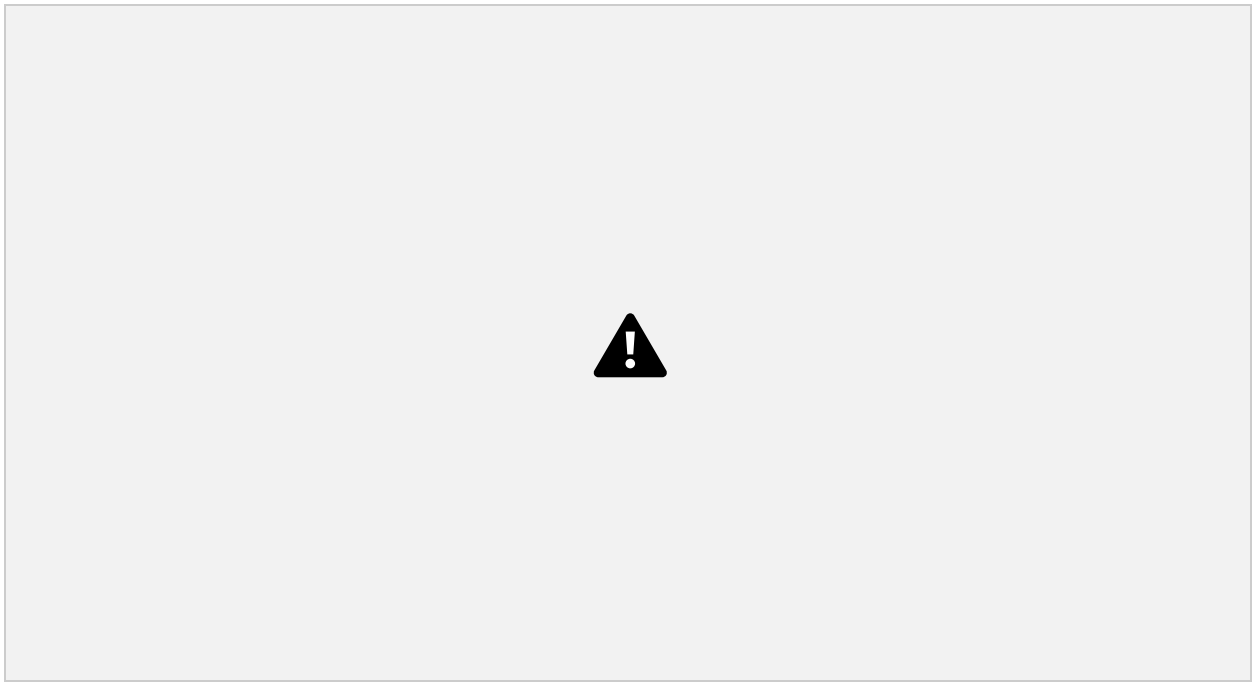
2.

Now you can run the server (note that you must have a JDK installed):

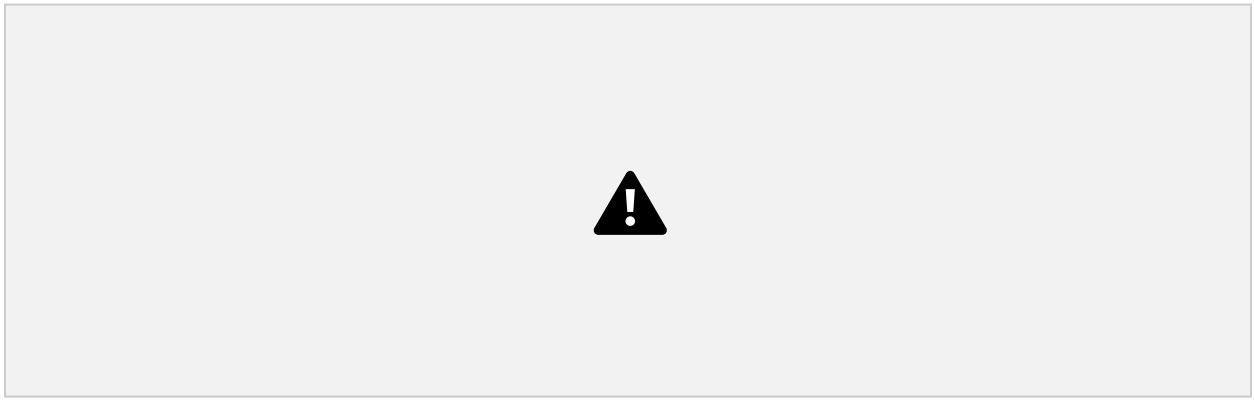
```
> ./fuseki-server
[2020-02-09 09:55:15] Server     INFO  Apache Jena Fuseki 3.17.0
[2020-02-09 09:55:15] Config     INFO
FUSEKI_HOME=/home/rene/fuseki/apache-jena-fuseki-3.17.0/.
[2020-02-09 09:55:15] Config     INFO
FUSEKI_BASE=/home/rene/fuseki/apache-jena-fuseki-3.17.0/run
[2020-02-09 09:55:15] Config     INFO  Shiro file:
file:///home/rene/fuseki/apache-jena-fuseki-3.17.0/run/shiro.ini
[2020-02-09 09:55:17] Server     INFO  Started 2020/02/09 09:55:17 EST on
port 3030
```

3.

As you can see, it starts a local server on port 3030, so open http://localhost:3030/ in your browser. You should see the main page:

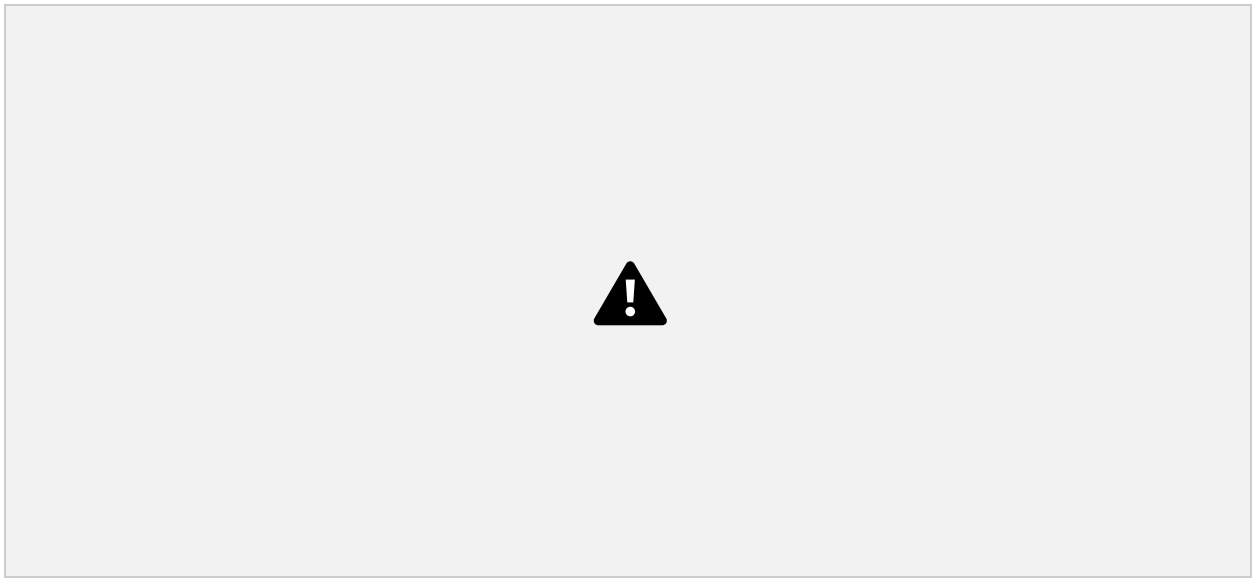


4. Before you can do anything else, you must create a *Dataset*, so select the first open above:

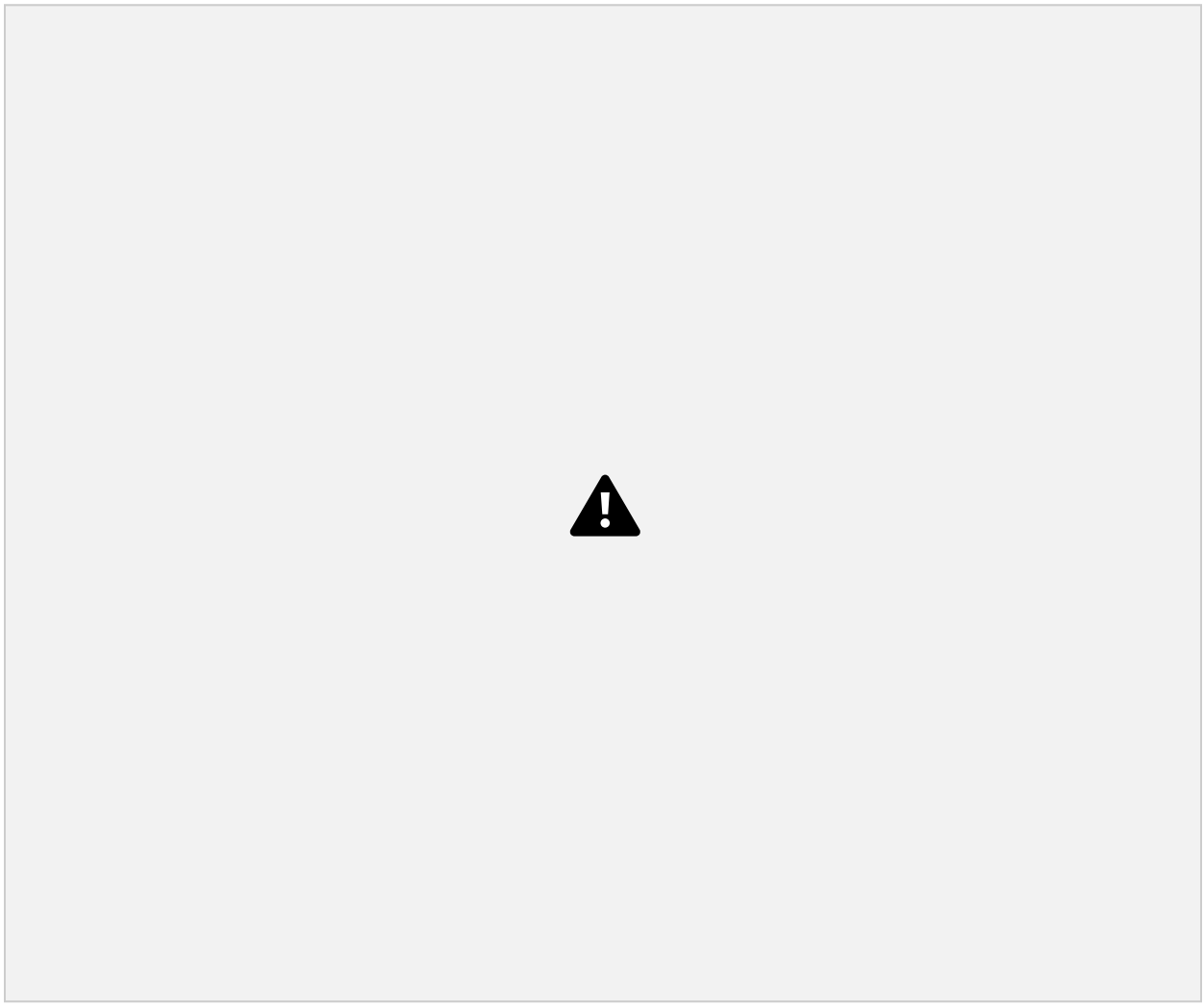


   Give your dataset a name and use the "In-memory" option (this means your triples will not be stored persistently, but this is fine for some first experiments).

5. You should now see your new dataset under "existing datasets" and can start to upload triples. Use your university (FOAF/FOCU) triples from two weeks ago (or any other triple file you have):



6. Now you are ready to send SPARQL queries to your server: go to the *query* tab and start querying your data:



Task #2.2: Learning SPARQL with Fuseki

With your own SPARQL server, you can now experiment with queries without relying on an external server. Go through the SPARQL Tutorial available on the Apache Jena, which comes with datasets and query exercises.

Task #3: SPARQL with Python

This task builds on the code you've previously developed with RDFlib.

Task #3.1: RDFlib

Continue the RDFlib introduction at Querying with SPARQL.

Task #3.2: Query University Data

Load some of the triples you created in the lab last week (FOAF/FOCU) into your program and query it using SPARQL. Write queries to:

1. List all students, sorted by age
2. Find all predicates and objects for a given student, searching by first name (e.g., *"Joe"*)
3. Print a count of all students by university

Task #3.3: Smart University Agent v1.1

Modify the code for your intelligent university agent from the previous week to use SPARQL queries, rather than manually traversing the graph.

That's all for this lab!