INM713 Semantic Web Technologies and Knowledge Graphs

Laboratory 10: `GraphDB`, an RDF Database for Knowledge Graphs

Ernesto Jiménez-Ruiz

Academic course: 2020-2021
Updated: March 30, 2021

Contents

1 Git Repositories **2**

2 Introduction **2**
2.1 Installation 2
2.2 Creating repositories 3

3 Loading data **4**
3.1 User interface 4
3.2 Programmatically 5

4 Querying the data **6**
4.1 User interface 6
4.2 Programmatically 7

5 Tasks **7**
5.1 Tasks with `GraphDB` graphical interface 7
5.2 Tasks with `GraphDB` as an Endpoint service 7
5.3 Optional tasks 7

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 Introduction

In the previous laboratory sessions we have used RDFLib (Python) and Jena (Java) to manage RDF graphs. These libraries are still useful; but to store, perform reasoning and query large knowledge graphs we need better solutions in the backend. RDFLib and Jena will still be used locally to, for example, create triples and communicate with a graph database via its SPARQL Endpoint.

In this module we are using the graph database `GraphDB`[1] developed by Ontotext (`https://www.ontotext.com/`) which has a set of interesting characteristics: *(i)* free version with a large set of features, *(ii)* compliant with Semantic Web standards, and *(iii)* with tutorials and documentation.[2]

2.1 Installation

`GraphDB` is available for basically all operating systems. To download the free version one needs to fill a form and then a link to the system is sent via email: `https://www.ontotext.com/products/graphdb/`.

For convenience we are using `GraphDB` as a desktop application. `GraphDB` can also be run as a command-line standalone server which is the standard way of using `GraphDB` in production. The Quick Start Guide (`https://graphdb.ontotext.com/documentation/free/quick-start-guide.html`) shows how to run `GraphDB` as a desktop application in the different platforms.[3] Once `GraphDB` has been launched via its Desktop application a window like the one in Figure 1 will appear. From this window you can change the settings of the `GraphDB` server (*e.g.*, the port,

[1] The screenshots are based on `GraphDB` version 9.6.

[2] `https://www.ontotext.com/knowledge-hub/`

[3] The desktop client, apart from running `GraphDB` as a server, also provides a user interface. In principle the desktop version is self-contained, but you may need to install Java 8: `https://www.java.com/en/download/`.

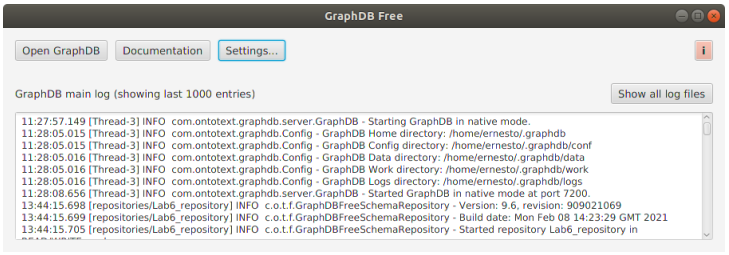

Figure 1: `GraphDB` initial window.

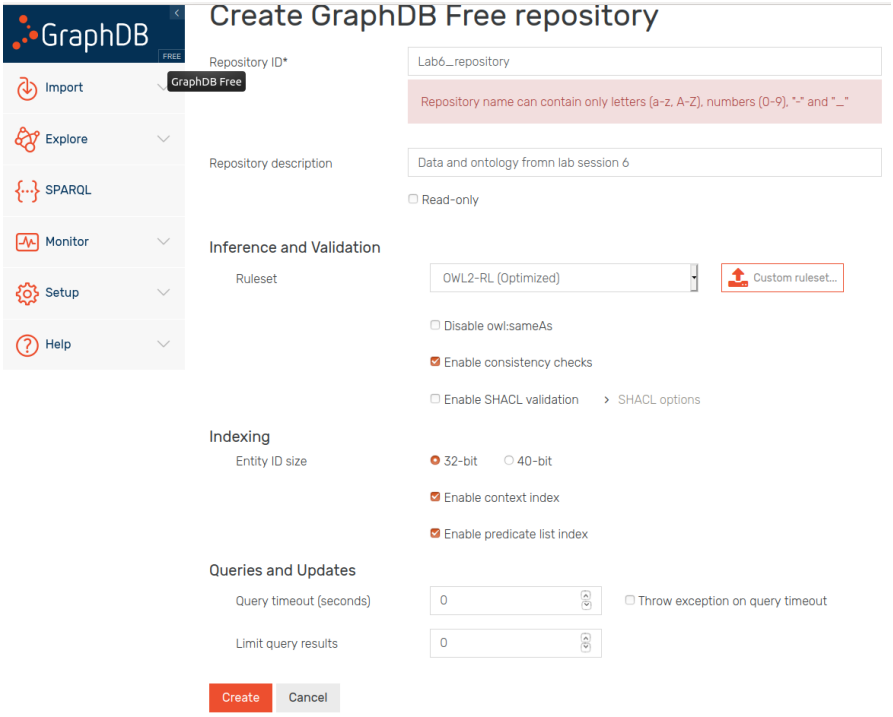

Figure 2: GraphDB repository creation.

the default is 7200), access the online documentation, or open GraphDB.[4] GraphDB opens via the (default) Web browser with the local URL `http://localhost:7200/`. You can use your favourite Web browser by just copying and pasting this GraphDB URL.

2.2 Creating repositories

The first step to start using GraphDB is the creation of a (GraphDB Free) repository with customised characteristics (see Figure 2). In our setting the most important ones are: *(i)* disabling Read-only (default), *(ii)* enabling OWL 2 RL reasoning from the list of supported languages, and *(iii)* enabling the consistency checks.

The created repository acts as a SPARQL Endpoint and can be accessed as such. Figure 3 shows how to access the URL of the created repository. For example:

`'http://192.168.0.18:7200/repositories/Lab6_repository'`

is the URL of the repository that has been created. It can be accessed programmatically as any other SPARQL Endpoint, as we covered during the laboratory session 3.

You can create several GraphDB repositories, although only one can be the *active* repository to be used with the desktop application (see GraphDB repositories view in Figure 4). Programmatically any of the repositories can be accessed, as long as GraphDB has been launched.

[4]Note that if port 7200 is already being used by another application, you will need to choose a different one.

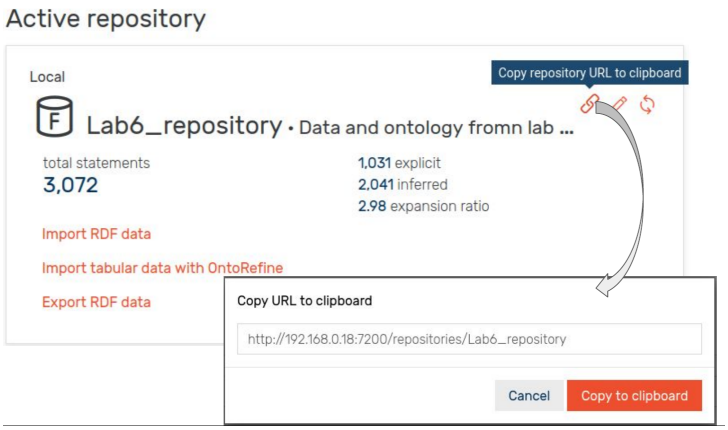

Figure 3: URL of the active repository in `GraphDB` (*e.g.*, URL of the SPARQL Endpoint).

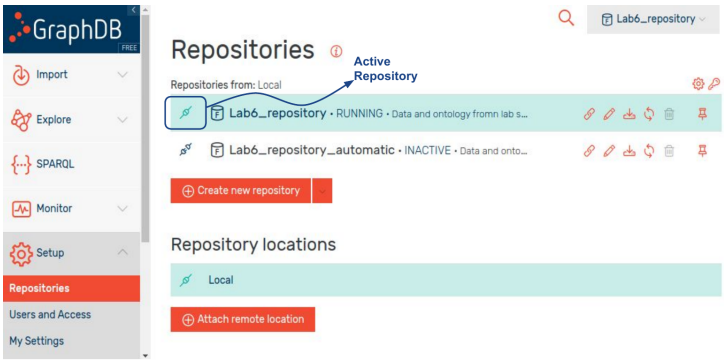

Figure 4: `GraphDB` repositories view.

3 Loading data

We can load data to a `GraphDB` repository via both the user interface and programmatically. We will use as example the data and ontology created in the laboratory session 6.

3.1 User interface

`GraphDB` allows to upload local and remote files in different formats including `.ttl` and `.owl` (see Figure 5). Once the files have been uploaded one need to import each of them into the repository (button *import*). You will be asked for a base URI, leave empty (will use the default in your data/ontology) or indicate a new one (*e.g.*, `http://www.semanticweb.org/ernesto/inm713/lab6/`). You will also be asked to add triples to the default graph or create a specific named graph.

One can also import RDF data and an ontology from a remote URL. For example you can use:

`https://raw.githubusercontent.com/city-knowledge-graphs/python/main/lab6/worldcities-free-100-task4.ttl`

`https://raw.githubusercontent.com/city-knowledge-graphs/python/main/lab6/ontology_lab6.owl`

Once the local files and/or remote resources have been loaded, you can explore the loaded triples in a tabular or graphical form (see Figure 6), and via SPARQL

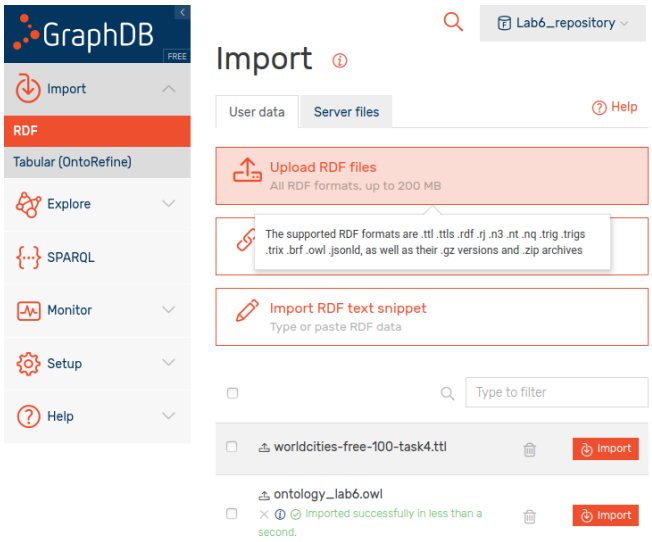

Figure 5: `GraphDB` upload and import RDF graphs and ontologies from files.

Figure 6: `GraphDB` RDF graph exploration.

queries (see Section 4). The data can also be cleared from the repository in the 'Graphs overview' menu.

3.2 Programmatically

There are several options to upload data programmatically.

- **SPARQL Update:**[5] although this has not been covered in the module the syntax is very similar to the SPARQL query language and may be useful to perform minor updates (*e.g.*, load individual triples).
- **LoadRDF tool:**[6] this is the best option for efficient (offline) upload of large amounts of data.
- **HTTP client request:** this is the option we will adopt in this lab session using the commandline tool `cURL` (Client URL).[7] `cURL` can be executed from Python

[5] `https://www.w3.org/TR/sparql11-update/`

[6] `https://graphdb.ontotext.com/documentation/free/loading-data-using-the-loadrdf-tool.html`

[7] Windows users may need to install `cURL`: `https://curl.se/windows/`

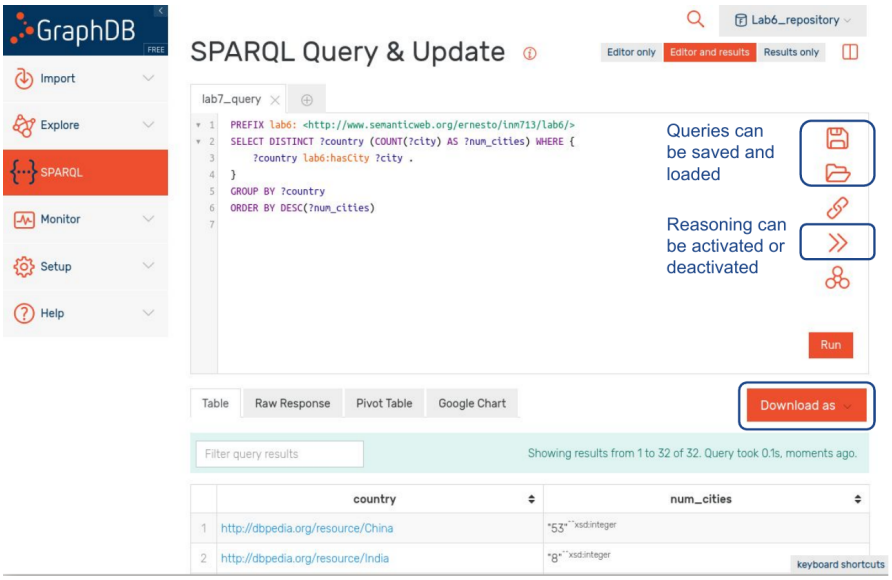

Figure 7: `GraphDB` query interface.

and Java. Command to be executed:

```
curl 'graphdb_upload_uri' -X POST -H
"Content-Type:application/x-turtle" -T 'datafile.ttl'
```

Where `'datafile.ttl` is the RDF data to load and `graphdb_upload_uri` is the concatenation of the SPARQL Endpoint URL (as in Figure 3) and `'/statements'`. For example:

```
http://192.168.0.18:7200/repositories/Lab6_repository/statements
```

Support codes: The Python and Java scripts `graphdb_communication.py` and `GraphdbCommunication.java`, respectively, provide an example of how to load into `GraphDB` the ontology and data from the Lab session 6.

4 Querying the data

We can query data from a `GraphDB` repository using SPARQL queries via both the user interface and programmatically.

4.1 User interface

Figure 7 shows the same SPARQL query as in Lab 7, Task 3.1 (see below). The query counts the cities in each country and gives the output ordered by number of cities. Using the `GraphDB` interface, *(i)* queries can be executed, *(ii)* queries can be named, saved and loaded, and *(iii)* query results can be downloaded in a number of formats (*e.g.*, CSV or JSON).

```
SELECT DISTINCT ?country (COUNT(?city) AS ?num_cities) WHERE {
    ?country lab6:hasCity ?city .
}
GROUP BY ?country
ORDER BY DESC(?num_cities)
```

4.2 Programmatically

The GraphDB repositories can be accessed and queried as a standard SPARQL Endpoint.

Support codes: Similarly to the solutions to the Lab session 3, the scripts `graphdb_communication.py` and `GraphdbCommunication.java` include an example to execute the above query over a GraphDB repository via its SPARQL Endpoint.

5 Tasks

Following the instruction above and the provided support codes complete the tasks below using GraphDB as a service and via its graphical user interface.

Task 1. Create two GraphDB repositories *e.g.*,: `'lab10_interface'` and `'lab10_programmatically'`.

5.1 Tasks with GraphDB graphical interface

Make sure that the `'lab10_interface'` repository is the active one.

Task 2. Upload the generated data and ontology created in the Lab session 6.

Task 3. Execute the query in Section 4.

5.2 Tasks with GraphDB as an Endpoint service

Use the `'lab10_programmatically'` repository from Python or Java. You will need the Endpoint URL of the repository as described in Section 2.2.

Task 4. Upload the generated data and ontology created in the Lab session 6 programmatically using the cURL command.

Task 5. Execute the query in Section 4 using the SPARQL Endpoint of the GraphDB repository.

5.3 Optional tasks

Task 6. Use GraphDB with the data and ontology generated as part of your coursework.

Task 7. `GraphDB` also includes a facility called `OntoRefine` (very similar to OpenRefine) to load data from CSV files and reconciling entities with Wikidata.[8] Load the CSV provided for the Lab session 6 using this facility. Try to generate similar triples as the model solution.

[8] Short tutorial video: `https://www.youtube.com/watch?v=YFb7hnZNLdQ`. Additional documentation: `https://graphdb.ontotext.com/documentation/free/loading-data-using-ontorefine.html`.