CITY
UNIVERSITY OF LONDON
EST 1894

IN3067/INM713 Semantic Web Technologies and Knowledge Graphs

Laboratory 7: SPARQL 1.1, GraphDB, Rules and SHACL

Ernesto Jiménez-Ruiz

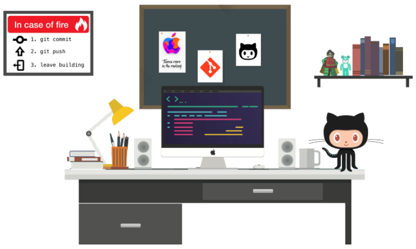
Academic course: 2023-2024
Updated: March 27, 2024

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`



2 SPARQL 1.1 queries

Create the following queries. Test them programmatically in Python or Java. Use the codes in the GitHub repositories (lab7) as example: `queryLocalRDFGraph.py` (or `lab7_sparql11_notebook.ipynb`) and `QueryLocalRDFGraph.java`

2.1 SPARQL Playground

We are using the SPARQL Playground (recall lab session 4). Use the `playground.ttl` data in the GitHub repositories.

Task 7.1 Get the number of people by sex.

Task 7.2 Select persons that DO NOT have any pets

Task 7.3 For each pet species get the number of pets and their average weight.

2.2 World cities dataset

Use the generated RDF graph from the World Cities dataset (lab session 5):

- Ontology: `ontology_lab5.ttl`
- Data: `worldcities-free-100-task2.ttl`

Task 7.4. Create a SPARQL query that counts the cities in each country. Order by number of cities.

2.3 Nobel prize dataset

Use the Nobel Prizes knowledge graph from the lab session 4:

- Ontology: `nobel-prize-ontology.rdf`
- Data: `nobelprize_kg.nt`

Tip: Performing reasoning with the Nobel prize knowledge graph takes some time. Using `GraphDB` (see Section 4), reasoning is much faster.

Task 7.5. Create a SPARQL query that returns countries with more than 10 Nobel laureates. Order by number of laureates.

Task 7.6. (Optional) Nobel Prize Laureates that are born in countries that have a population smaller than $1,000,000$ according to DBpedia. This requires the use of the SPARQL 1.1 SERVICE functionality to connect to DBpedia.
Note: *To test this query programmatically, I could only make this type of queries (using federation) work with Jena. It can also be tested using `GraphDB`, see Section 4.5.3.*

2.4 DBpedia knowledge graph

Task 7.7. Test the queries in the lecture slides using the DBPedia SPARQL Endpoint: `https://dbpedia.org/sparql`.

3 SPARQL 1.1 Update

SPARQL 1.1 UPDATE is a language to modify a given RDF graph. The codes in the respective GitHub repositories (*i.e.*, `sparqlUpdateExample.py`, `lab7_sparql11_notebook.ipynb` and `SparqlUpdateExample.java`) provide some examples to insert and delete triples with the SPARQL Playground dataset.

Task 7.8. Create a SPARQL Update to insert information about your pets. If you do not have a pet create a fictional one.

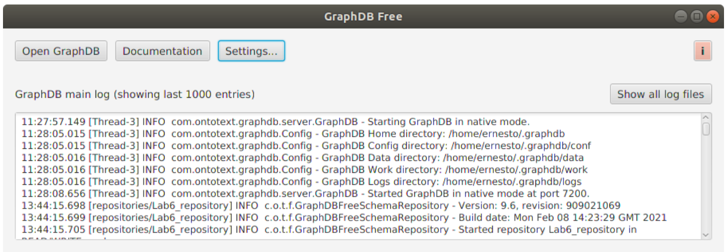
4 GraphDB tutorial

[To be started in the lab and completed at home].

In the 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:

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



GraphDB main log (showing last 1000 entries)

```
11:27:57.149 [Thread-3] INFO  com.ontotext.graphdb.server.GraphDB - Starting GraphDB in native mode.
11:28:05.015 [Thread-3] INFO  com.ontotext.graphdb.Config - GraphDB Home directory: /home/ernesto/.graphdb
11:28:05.015 [Thread-3] INFO  com.ontotext.graphdb.Config - GraphDB Config directory: /home/ernesto/.graphdb/conf
11:28:05.016 [Thread-3] INFO  com.ontotext.graphdb.Config - GraphDB Data directory: /home/ernesto/.graphdb/data
11:28:05.016 [Thread-3] INFO  com.ontotext.graphdb.Config - GraphDB Work directory: /home/ernesto/.graphdb/work
11:28:05.016 [Thread-3] INFO  com.ontotext.graphdb.Config - GraphDB Logs directory: /home/ernesto/.graphdb/logs
11:28:08.656 [Thread-3] INFO  com.ontotext.graphdb.server.GraphDB - Started GraphDB in native mode at port 7200.
13:44:15.698 [repositories/Lab6_repository] INFO  c.o.t.f.GraphDBFreeSchemaRepository - Version: 9.6, revision: 909021069
13:44:15.699 [repositories/Lab6_repository] INFO  c.o.t.f.GraphDBFreeSchemaRepository - Build date: Mon Feb 08 14:23:29 GMT 2021
13:44:15.705 [repositories/Lab6_repository] INFO  c.o.t.f.GraphDBFreeSchemaRepository - Started repository Lab6_repository in
```

Figure 1: GraphDB initial window.

(i) free version with a large set of features, *(ii)* compliant with Semantic Web standards, and *(iii)* with tutorials and documentation.[2]

4.1 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 Getting Started Guide (https://graphdb.ontotext.com/documentation/10.2/getting-started.html) shows how to install and 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, 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.

4.2 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.

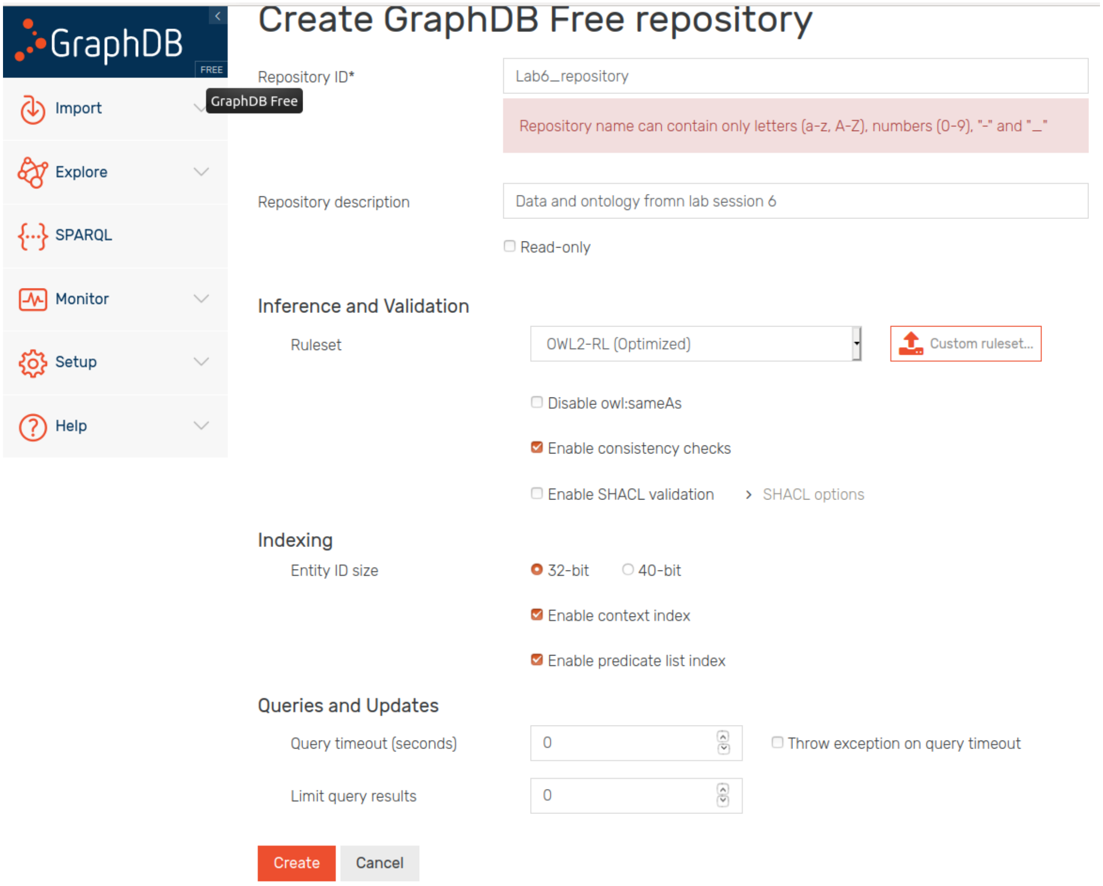
The created repository acts as a SPARQL Endpoint and can be accessed as such. Figure 3 shows how to access the URL of the created repository. For example:

`'http://127.0.0.1:7200/repositories/lab_graphdb'`

[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/.

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



Create GraphDB Free repository

Repository ID* Lab6_repository

Repository name can contain only letters (a-z, A-Z), numbers (0-9), "-" and "_"

Repository description Data and ontology fromn lab session 6

□ Read-only

Inference and Validation

Ruleset OWL2-RL (Optimized) Custom ruleset...

□ Disable owl:sameAs

☑ Enable consistency checks

□ Enable SHACL validation › SHACL options

Indexing

Entity ID size ● 32-bit ○ 40-bit

☑ Enable context index

☑ Enable predicate list index

Queries and Updates

Query timeout (seconds) 0 □ Throw exception on query timeout

Limit query results 0

Create Cancel

Figure 2: `GraphDB` repository creation.

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 4.

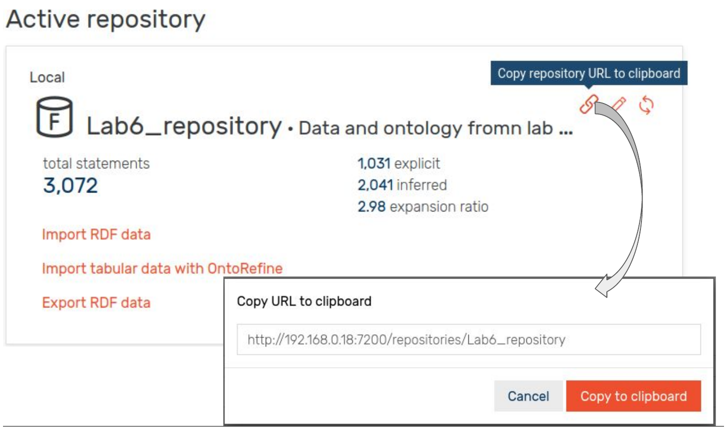
You can create several `GraphDB` repositories, although only one can be the *active* repository to be used with the desktop application (see `GraphDB` repositories view in Figure 4). Programmatically any of the repositories can be accessed, as long as `GraphDB` has been launched.

4.3 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 (World cities dataset) created in the laboratory session 5.

4.3.1 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. You will also be asked to add triples to the default graph or create a specific named graph.



Lab6_repository • Data and ontology fromn lab ...

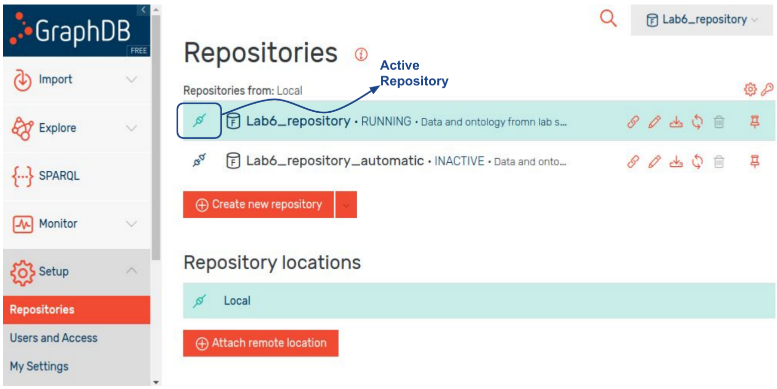
total statements 3,072

1,031 explicit
2,041 inferred
2.98 expansion ratio

Copy URL to clipboard

http://192.168.0.18:7200/repositories/Lab6_repository

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



Repositories from: Local

Lab6_repository • RUNNING • Data and ontology fromn lab s...

Lab6_repository_automatic • INACTIVE • Data and onto...

Local

Figure 4: `GraphDB` repositories view.

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-2024/main/lab7/data/worldcities-free-100-task2.ttl`
- `https://raw.githubusercontent.com/city-knowledge-graphs/python-2024/main/lab7/data/ontology_lab5.ttl`

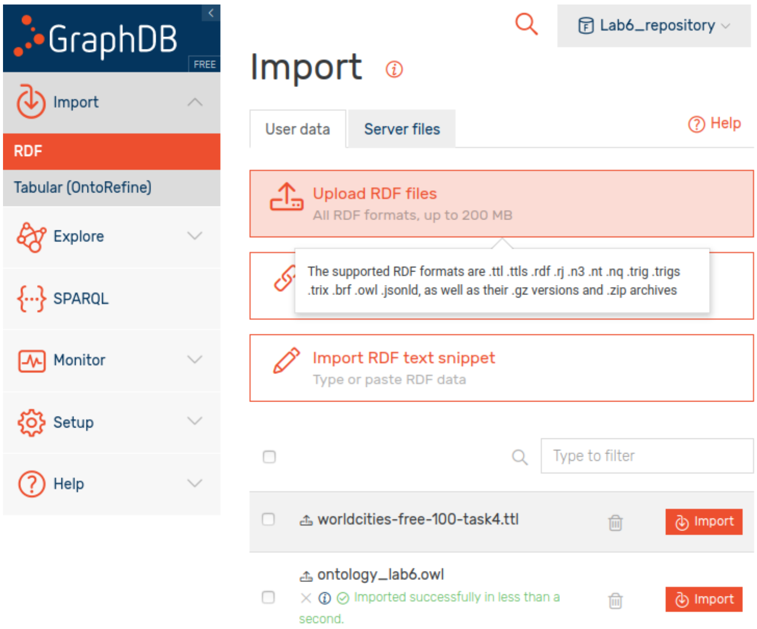
Once the local files and/or remote resources have been loaded, you can explore the loaded triples in a tabular or graphical form (see Figure 6), and via SPARQL queries (see Section 4.4). The data can also be cleared from the repository in the 'Graphs overview' menu.

4.3.2 Programmatically

There are several options to upload data programmatically.

- **SPARQL Update:**[5] using the SPARQL Update language may be useful to perform minor updates (*e.g.*, load individual triples).

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

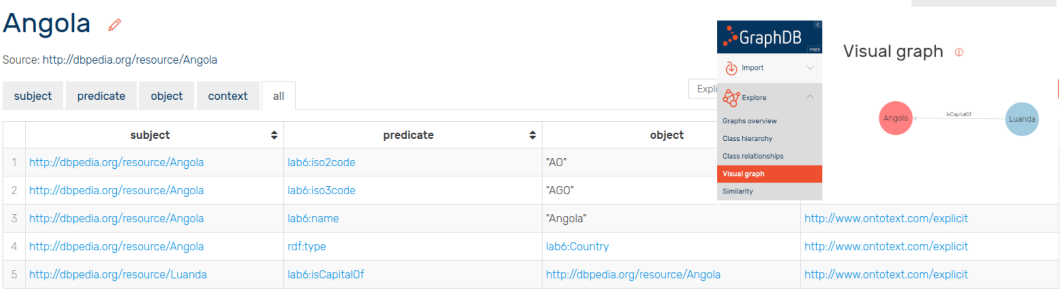


The supported RDF formats are .ttl .ttls .rdf .rj .n3 .nt .nq .trig .trigs .trix .brf .owl .jsonld, as well as their .gz versions and .zip archives

□ worldcities-free-100-task4.ttl

□ ontology_lab6.owl
Imported successfully in less than a second.

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



Angola

Source: http://dbpedia.org/resource/Angola

	subject	predicate	object	
1	http://dbpedia.org/resource/Angola	lab6:iso2code	"AO"	
2	http://dbpedia.org/resource/Angola	lab6:iso3code	"AGO"	
3	http://dbpedia.org/resource/Angola	lab6:name	"Angola"	http://www.ontotext.com/explicit
4	http://dbpedia.org/resource/Angola	rdf:type	lab6:Country	http://www.ontotext.com/explicit
5	http://dbpedia.org/resource/Luanda	lab6:isCapitalOf	http://dbpedia.org/resource/Angola	http://www.ontotext.com/explicit

Figure 6: `GraphDB` RDF graph exploration.

- **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 and Java. Command to be executed:

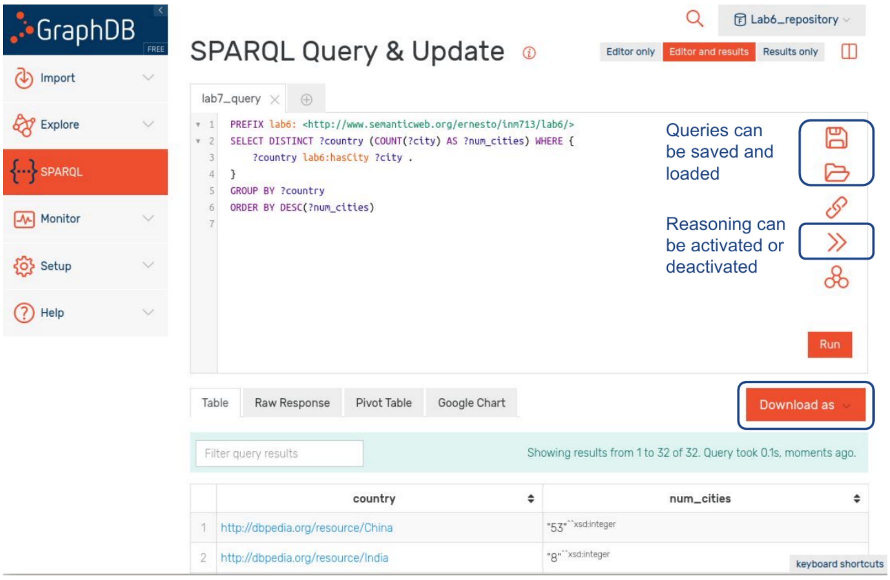
  ```
  curl ‘graphdb_upload_uri’ -X POST -H
  “Content-Type:application/x-turtle” -T ‘datafile.ttl’
  ```

  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://127.0.0.1:7200/repositories/lab_graphdb/statements
```

[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/`



```
PREFIX lab6: <http://www.semanticweb.org/ernesto/inm713/lab6/>
SELECT DISTINCT ?country (COUNT(?city) AS ?num_cities) WHERE {
    ?country lab6:hasCity ?city .
}
GROUP BY ?country
ORDER BY DESC(?num_cities)
```

Showing results from 1 to 32 of 32. Query took 0.1s, moments ago.

	country	num_cities
1	http://dbpedia.org/resource/China	"53"^^xsd:integer
2	http://dbpedia.org/resource/India	"8"^^xsd:integer

Figure 7: `GraphDB` query interface.

Support codes: The Python and Java scripts `graphdb_communication.py` and `GraphdbCommunication.java`, respectively, provide examples of how to load in and query from `GraphDB`.

4.4 Querying the data

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

4.4.1 User interface

Figure 7 shows a query that 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 lab5:hasCity ?city .
}
GROUP BY ?country
ORDER BY DESC(?num_cities)
```

4.4.2 Programmatically

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

Support codes: Similarly to the solutions to the Lab session 4, the scripts `graphdb_communication.py` and `GraphdbCommunication.java` include an example to execute the above query over a GraphDB repository via its SPARQL Endpoint.

4.5 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 7.9. Create two GraphDB repositories *e.g.*,: 'lab7_interface' and 'lab7_programmatically'.

4.5.1 Tasks with GraphDB graphical interface

Make sure that the 'lab7_interface' repository is the active one.

Task 7.10. Upload the generated data and ontology created in the Lab session 5.

Task 7.11. Execute the query in Section 4.4.

4.5.2 Tasks with GraphDB as an Endpoint service

Use the 'lab7_programmatically' repository from Python or Java. You will need the Endpoint URL of the repository as described in Section 4.2.

Task 7.12. Upload the generated data and ontology created in the Lab session 5 programmatically using the cURL command (see support codes).

Task 7.13. Execute the query in Section 4.4 using the SPARQL Endpoint of the GraphDB repository.

4.5.3 GraphDB with the Nobel Prize dataset.

Create a new GraphDB repository for the Nobel Prize dataset. Use the scripts graphdb_communication.py and GraphdbCommunication.java as example to load and query the dataset. Note that reasoning is much faster with GraphDB.

Task 7.14. Run the query from Task 7.5 over the created GraphDB repository.

Task 7.15. Try to execute the queries using federation[8] (*i.e.*, SPARQL SERVICE functionality) from the GraphDB interface (*e.g.*, query in Task 7.6 and query query_nobel-prize-service.txt in the GitHub repository)

4.5.4 GraphDB with named graphs

RDFlib and Jena do not provide a clear support for named graphs. GraphDB, however, implements this functionality.[9]

[8] https://graphdb.ontotext.com/documentation/standard/sparql-federation.html

[9] https://graphdb.ontotext.com/documentation/standard/query-behaviour.html

Task 7.16. Create a new `GraphDB` repository to store the named graphs in `named_graphs.ttl`. Note that the format of the file is *TriG* as regular Turtle do not support named graphs. Extend the `named_graphs.ttl` dataset to include triples about yourself in the correspondent named graph. Use the scripts `graphdb_communication.py` and `GraphdbCommunication.java` as example to load and query the named graphs.

5 Protégé tutorial: SPARQL, Rules and SHACL

(Optional) From the Pizza tutorial: `https://www.michaeldebellis.com/post/new-protege-pizza-tutorial`.

Task 7.17 Follow Chapter 8 to add instances in the Pizza OWL ontology. This is required for the following tasks.

Task 7.18 Follow Chapter 9 to create SPARQL queries in Protégé. Note that this SPARQL functionality does not make use of the reasoner. but it may be useful to test simple queries over the ontology plus the generated data in Part 2 of the coursework.

Task 7.19 Follow Chapter 10 to create SWRL rules in Protégé. You will need to install the SWRL plugin (`SWRLTab`).

Task 7.20 Follow Chapter 11 to create SHACL constraints in Protégé. You will need to install the SHACL plugin (`SHACL4Protege`).

6 Solutions

Check GitHub to test the queries programmatically.

6.1 SPARQL Task 7.1

```
SELECT DISTINCT ?sex (COUNT(?people) as ?peopleCount) WHERE {
    ?people rdf:type dbo:Person .
    ?people tto:sex ?sex .
}
GROUP BY ?sex
```

6.2 SPARQL Task 7.2

```
SELECT DISTINCT ?person ?pet WHERE {
    ?person rdf:type dbo:Person .
    FILTER NOT EXISTS {?person tto:pet ?pet }.
}
```

6.3 SPARQL Task 7.3

```
SELECT DISTINCT ?species (COUNT(?pet) as ?petCount) (AVG(?weight)
as ?avgWeight) WHERE {
    ?species rdfs:subClassOf tto:Animal .
    ?pet rdf:type ?species .
    ?pet tto:weight ?weight .
}
GROUP BY ?species
```

6.4 SPARQL Task 7.4

```
SELECT DISTINCT ?country (COUNT(?city) AS ?num_cities) WHERE {
    ?country lab5:hasCity ?city .
}
GROUP BY ?country
ORDER BY DESC(?num_cities)
```

6.5 SPARQL Task 7.5

```
SELECT DISTINCT ?country (COUNT(?laur) AS ?num_laur) WHERE {
    ?laur rdf:type nobel:Laureate .
    ?laur dbpedia-owl:birthPlace ?country .
    ?country rdf:type dbpedia-owl:Country .
}
GROUP BY ?country
HAVING (COUNT(?laur) > 10)
ORDER BY DESC(?num_laur)
```

6.6 SPARQL Task 7.6

```
SELECT DISTINCT ?label ?country WHERE {
    ?laur rdf:type np:Laureate .
    ?laur rdfs:label ?label .
    ?laur dbo:birthPlace ?country .
    ?country rdf:type dbo:Country .
    ?country owl:sameAs ?dbr .
    SERVICE <http://dbpedia.org/sparql> {
      ?dbr dbo:populationTotal ?pop .
      FILTER (?pop < 1000000)
    }
}
```