LAB Assignment

This is a hands-on lab on knowledge graphs[1]*. We will use the GraphDB database for working with knowledge graphs. To prepare for the session, you are recommended to get familiar with the RDF Schema vocabulary specification, SPARQL specification and GraphDB manual.*
In the sequel, we provide the environment setup and then exercises to be solved. Each group must upload the solutions to Learn-SQL (look for the corresponding assignment event). One group member must submit the solutions (check below for a precise enumeration of what you need to submit) and list all group members in such document. Check the assignment deadline and be sure to meet it. It is a strict deadline!

What to deliver?

Upload ONE compressed file with your solutions. The main file of your solution must be a pdf file named as '`[Group]-[MemberSurname]+.pdf`'. The file must be structured according to the sections you will find below and, within each section, specify your solution following the instructions of each section. If you make any assumption not explicit in the statement, add a note in the corresponding section. Some sections ask you to deliver code. In those cases, you must attach a file per section named as follows: '`[Group]-(SUB)SECTION-[MemberSurname]+`'.

Setup Instructions

For section A

For section A, you need to load an excerpt of the DBpedia TBOX and ABOX to GraphDB. We will use DBpedia to get used to GraphDB. Follow these steps:

- Download the Graph DB installer: `https://www.ontotext.com/products/graphdb/graphdb-free/`
  Fill in the form that appears on the right. After submitting the form, you will receive an email at the address you provided with various download links depending on your OS (Windows, MacOS or Linux). Please also check your junk / spam folder in case the email went there.

- Install GraphDB: After downloading the installer, you should run normally and you do not require an administrator account to run it. Follow the steps that appear in the executor without changing any of the configurations and wait for the installation process to finish.

[1] This statement distinguishes the concepts of knowledge graph and ontologies. Refer to the main lectures to know the difference, but, in short, an ontology is a knowledge graph distinguishing the TBOX and ABOX and using inference. Be thoughtful about the wording of the exercise statements and their usage.

LAB Assignment

- Launch GraphDB: Once the installation is complete, you should start the GraphDB server by launching the `'GraphDB Free'` application from your home / start menu. Once the server is up, a new window in your browser should open automatically with the URL `http://localhost:7200/`.

- Configure settings: Go to the GraphDB application (i.e., the background stand-alone window) and click on the button `'Settings...'` at the top. A new pop-up window should open, and you should introduce the following custom java properties (the first one sets the maximum number of triples that can be loaded via GraphDB Workbench and the heap space used by the JVM):

  `graphdb.workbench.maxUploadSize=40000000000`

  `Xmx=2000m`

  Afterwards, click on the `'Save and restart'` button at the bottom right. The GraphDB server will restart and will open a new browser window again with the same URL as above.

- Load a dump from DBpedia: It includes both the DBpedia TBOX and **some** ABOX instances. Download the two files from the following URLs (note that the second file is large and might take a few minutes to download):

  `http://downloads.dbpedia.org/2014/dbpedia_2014.owl.bz2`

  `http://downloads.dbpedia.org/2014/en/instance_types_en.nt.bz2`

  Once the download is complete, unzip both downloaded files. This should generate two files, one with the extension `.owl` and another with the extension `.nt`. In the knowledge graph world, the extension of the file refers to the format used to express the file triples.

- Create a repository: Before importing any data, we need to first create a repository. Go to the GraphDB interface (i.e., to the browser tab) and select `'Setup'` and then `'Repositories'` from the left-hand menu. Afterwards, click on the `'Create new repository'` and apply the following steps:

  Insert a `'Repository ID'` and `'Repository title'` of your own.

  Turn inference on: Under `'Ruleset'` select `'RDFS-Plus (Optimized)'`. This tells GraphDB to activate reasoning. The option chosen tells GraphDB to perform reasoning based on an optimized version of the RDFS regime entailment inference rules.

  Note: GraphDB supports inference out of the box (check the theoretical slides for more information). Inference is the derivation of new knowledge from existing knowledge and axioms. It is used for deducing further knowledge based on existing RDF data and a formal set of inference rules. To understand how GraphDB deals

with reasoning please check:

`http://graphdb.ontotext.com/documentation/free/inference.html#inference-in-graphdb`,
`http://graphdb.ontotext.com/documentation/standard/reasoning.html` and
`https://graphdb.ontotext.com/documentation/free/rdfs-and-owl-support-optimisations.html`
The first link explains how inference can be customized in GraphDB. The second one explains the reasoning process and how GraphDB materializes it. The third explains the difference between the RDFS inference rules seen in the lectures and what GraphDB calls the optimized version of them.

- Load data into the repository: Go the GraphDB interface and select `'Import'` and then `'RDF'` from the left-hand menu. Afterwards, click on the `'Upload RDF files'` button and choose the two files that you just extracted (uncompress them first!). Wait for uploading the files to be finished, it should take around 3 - 4 minutes to upload the larger file. Once both files are uploaded, select them both and click on the `'Import'` button at the top. A new pop-up will appear and you need to fill in the following configuration parameters:

  Base IRI: enter `http://dbpedia.org/resource/`

  Target graphs: choose `'Named graph'` and enter the following in the text box: `http://localhost:7200/dbpedia/`

  After that, click on the `'Import'` button at the bottom right corner of the pop-up and wait for the import operation to finish. Since the files are large, the import will take around 10 minutes to be done.

- Test: Go to `'SPARQL'` on the left-hand menu and copy the following query into the text box that appears:

  ```
  SELECT DISTINCT ?s WHERE { ?s ?p ?o .  } LIMIT 50
  ```

  Click on the `'Run'` button at the bottom right corner of the text box to execute the query. If the query does not return any results, this means that no data was found in the repository and that the import operation failed. If it does return results, the import was successful and you can proceed with the exercises of the session.

- Turn on autocomplete: Click on `'Setup'` and then `'Autocomplete'` from the left-hand menu. In the page that appears, toggle the button at the top under `'Autocomplete index'` to change it from off to on. A new status that says `'building'` should appear to the right of the toggle. This will activate autocomplete when we are executing queries and will allow us to search for properties or classes that are defined in our knowledge

graph. This feature, similar to that in most programming IDEs, will help you massively during the lab. You do not need to wait for the building to finish, you can continue with the exercises of this session and the autocomplete index building will continue in the background.

Important note: all the above steps are meant to facilitate your first steps as an initial walkthrough to GraphDB. Remember though you are responsible to learn the specificities of GraphDB on your own if the above explanations are not enough, so be sure you understand what each step does. Also, you may want to activate other options that you may find interesting.

A Exploring DBpedia

DBpedia

DBpedia[2] *is a crowd-sourced community effort to extract structured content from the information created in various Wikimedia projects. This structured information resembles an open knowledge graph which is available for everyone on the Web. A knowledge graph is a special kind of database which stores knowledge in a machine-readable form and provides means for information to be collected, organised, shared, searched and utilised. Google uses a similar approach to create those knowledge cards during search.*

Tasks

One of the main challenges of a knowledge graph with regard to a traditional (e.g., relational) database is that we may not know, at all, the terminological axioms (i.e., the TBOX) followed by data instances (i.e., the ABOX). Therefore, the first usual step to start working with a knowledge graph is to **explore** its TBOX. If you remember, during the lectures, at home, you had to solve a SPARQL exercise that left you alone with the burden of exploring DBPedia. There, you had to write basic SPARQL queries to find classes, properties and, once you identified an interesting one, how to find triples where it participates. All in all, that exercise guide you, for first time, through an exploratory effort over a knowledge graph.

The advantage of using GraphDB, however, is that it offers an advanced visual interface to explore the classes defined in a knowledge graph that saves us from executing a manual exploratory exercise. Indeed, GraphDB provides nice add-ons to explore the knowledge graph in a more efficient way. To access these features, click on `'Explore'` from the left-hand menu and browse the different options. The most relevant one, probably, is `'Class hierarchy'`. Click on it and wait a few seconds for the graph to load. After that, you should get a visual graph that depicts the different classes defined in the DBpedia knowledge

[2] https://wiki.dbpedia.org

graph and the hierarchies between them. For instance, you can see that the biggest class is called `'Agent'`, which has many sub-classes. When clicking on the bubble representing each class, a pop-up panel with metadata about that class appears on the right. This panel contains its semantic bag (i.e., related properties and adjacent classes), the name of the class, its description (obtained through the RDFS property rdfs:label used to provide a free-text description for human consumption) and some of its instances. Note the semantic bag concept (retrieved through the domain-range graph) is similar to the DESCRIBE clause in SPARQL and therefore meets the requiements to *derefer* a URI). Also, realize that the results retrieved by GraphDB through these features **are affected by the inference capabilities selected when creating the repository**.

Spend some minutes exploring DBPedia, both its classes and properties, and browse around and understand all explorative options provided by GraphDB. Once your curiosity is satisfied, go back to the `'SPARQL'` page and execute queries that are related to the classes and properties defined by DBpedia. This exercise is meant to be exploratory and to let you get familiar with GraphDB. There is no delivery associated to this section.

Note: When querying, it is important to note that the semantics of the knowledge graph are determined by different namespaces, which are defined by the "prefix" keyword at the beginning of the query. Thus, it is important to properly understand and use them. Also, be aware of the inference capabilities activated when answering the questions above. Finally, in the SPARQL page, be sure the option "*include inferred data in results*" is ON when using inference.

B Ontology creation

Let's build an ontology. Creating an ontology is not simple and requires training. Thus, in this section we will train your modeling skills as well as introduce you to the required tools to manipulate and create triples. In this assignment we will practice how to create and query your own **ontology**.

B.1 TBOX definition

Define a TBOX for the research publication domain following the idea of Lab 1 (this way, this will also help you to implicitly get a grasp of pros and cons with regard to property graphs). Thus, we want to model the concepts of paper, authorship, publication and review. Specifically, authors write research papers that can be published in the proceedings of a conference or workshop (a conference is a well-established forum while a workshop is typically associated to new trends still being explored), or in a journal. A conference/workshop is organized in terms of editions. Each edition of a conference/workshop is held in a given city (venue) at a specific period of time of a given year. Proceedings are published records which include all the papers presented in an edition of a conference/workshop. Oppositely, journals

do not hold joint meeting events and, like a magazine, a journal publishes accepted papers in terms of volumes. There can be various volumes of a journal per year. A paper can be written by many authors, however only one of them acts as corresponding author. A paper can be cited by another paper, meaning their content is related. A paper can be about one or more topics, specified by means of keywords (e.g., property graph, graph processing, data quality, etc.). A paper must also contain an abstract (i.e., a summary of its content). Finally, we also want to include in the graph the concept of review. When a paper is submitted to a conference/workshop or a journal, the conference chair or the journal editor assigns a set of reviewers (typically three) to each paper. Reviewers are scientists and therefore they are relevant authors (i.e., they have published papers in relevant conferences or journals). Obviously, the author of a certain paper cannot be reviewer of her own paper.

Unfortunately, GraphDB does not provide any feature to create a TBOX / ABOX beyond SPARQL CONSTRUCT. To create the TBOX, you have two options: either create them directly via SPARQL CONSTRUCT or use an external tool for that purpose. For the latter, there are two options: tools providing an interface to create triples, such as Protégé[3] or VocBench[4], or programmatic tools (such as Jena or RDFLib -see the ABOX definition section to know more about programmatic tools-). One relevant aspect of this exercise is to decide what knowledge graph language to use: RDF, RDFS or OWL.

Tasks:

1. Independently of how you created the TBOX, you need to provide either:

   - the SPARQL queries you used for creating the TBOX (in case you used SPARQL CONSTRUCT). In this case, add them to the corresponding section in the main deliverable PDF file and, importantly, include a graphical representation (e.g., using `https://gra.fo/`).
   - in case you used a visual tool (e.g., Protégé or VocBench) include a graphical representation in the pdf file (e.g., using `https://gra.fo/` or built-in services in those tools) and include the rdf, rdfs or owl file generated by the tool in the zip file to upload and following the naming notation explained for the deliverables (**IMPORTANT**: the lecturer should not install any additional tool to validate this part).
   - If you generated the TBOX in a programmatic manner, you still need to provide a graphical representation in the main pdf file, the output rdf, rdfs or owl file **and** additionally the code to generate the TBOX. The additional files must follow the naming notation explained.

[3] `https://protege.stanford.edu`
[4] `http://vocbench.uniroma2.it/downloads`

B.2 ABOX definition

In this section we want to create an ABOX compliant with the TBOX above created. For this exercise we strongly advise you to reuse the data you built for the research publications domain from the *Assignment of Lab 1* and adapt it to this exercise.

Similar to the property graph lab, it is usual to convert CSV, JSON or even relational data into RDF, RDFS or OWL. The Knowledge Graph community has been very active creating tools to generate triples from any other source. For example, two prominent projects are Any23[5] and Open Refine[6] with its RDF extension[7] (which can directly connect to GraphDB). Nevertheless, it is highly unlikely that simply by using these tools you are able to generate the required ABOX. Most probably, you will need a framework to further create and manipulate triples. For this purpose, Jena API is the most prominent and popular resource[8]. Jena is a powerful framework whose main components are:

- RDF API: it is the core API to create and manipulate triples. Use it together with the ontology API to create RDFS or OWL ontologies. Importantly, once created, it allows to serialize the triples in different formats (e.g., turtle, N3, etc.). In case you want to perform reasoning by means of any regime entailment inference, you will need the inference API. There is a very comprehensive tutorial about the RDF API at: `https://jena.apache.org/tutorials/rdf_api.html`.
- TDB: you can store your triples in an external database (e.g., GraphDB) or you can use its internal TDB. For this exercise, you are advised to better use GraphDB and connect from Jena to it to save your created TBOX and ABOX.
- Finally, Jena also provides ARQ to query your data programmatically via SPARQL.

In principle, you will not need TBD and ARQ for this exercise, since as explained later, we advise you loading your TBOX and ABOX files into GraphDB and query it from there (via the import option you practiced in Section A).

Note: Jena is a Java API. In the meantime, other solutions have emerged to work with knowledge graphs in other languages. In Python the most popular one is, probably, RDFLib[9]. It is still not as mature as Jena, but it has massively improved in the last years. You are free to use one or another but before starting.

[5] `https://any23.apache.org`
[6] `http://openrefine.org/`
[7] `https://github.com/stkenny/grefine-rdf-extension/wiki`
[8] `http://jena.apache.org/index.html`
[9] `https://rdflib.readthedocs.io/en/stable/`

Tasks

1. In the main pdf file, explain the methodology used to define your ABOX from non-semantic data.
2. In the zip file, include your code to create the ABOX programmatically (e.g., Jena or RDFLib code) and the resulting ABOX file (rdf, rdfs or owl). These files must follow the naming notation explained.

B.3 Create the final ontology

Import your TBOX and ABOX files to GraphDB. When loading the data, you are also required to link the ABOX with the TBOX via rdf:type. Be aware, however, that depending on the inference regime entailment you consider for your exercise, you may save generating quite a few of them.

Note: Even if it is not part of this lab and you are not asked to do it, be aware that one relevant aspect of creating knowledge graphs is linking it to external knowledge graphs.

Tasks

You have two options for linking the TBOX and the ABOX. Choose one of them:

1. Once loaded in GraphDB, provide the SPARQL CONSTRUCT queries required to create the link between the ABOX and TBOX (not recommended).
2. Create the links programmatically (e.g., in Jena or RDFLib) before loading them in GraphDB. In case you go for this solution, the code for creating the links and the additional file (rdf, rdfs or owl) generated must be provided and named according to the naming convention explained (recommended).

Besides that, you need to provide two additional outputs for this section (both must be included in the main pdf file of the deliverable):

- Specify the inference regime entailment you are considering. Explain what rdf:type links you saved to explicitly generate thanks to the inference rules.
- Provide a summary table of your instances. Compute simple statistics about the resulting knowledge graph. For example, the number of classes, the number of properties, number of instances for the main classes and number of triples using the main properties.

B.4 Querying the ontology

Once the ontology has been created and loaded into GraphDB it is ready to be queried. We are specifically interested in the benefits of having an explicit TBOX defined and enabling reasoning.

Tasks

In the main pdf file, provide the following SPARQL queries (explicitly state any assumption you make):

1. Find all Authors.
2. Find all properties whose domain is Author.
3. Find all properties whose domain is either Conference or Journal.
4. Find all the papers written by a given author that where published in database conferences.

Finally, create two additional SPARQL queries. We will evaluate how interesting the queries are and how interesting is the retrieved information. Show us you know how to squeeze a Knowledge Graph!