This is a hands-on lab on Property Graphs. We will be using what is, most probably, the most popular property graph database: Neo4j. We will practice how to create and manage graphs, as well as querying/processing them.

In the sequel, we provide the instructions for setting up the environment and then we list the exercises to be solved. One group member, in the name of the group, must upload the solution to the Learn-SQL platform, but remember to include the name of all group members in your solutions. Please check the assignment deadline and be sure to meet it. It is a strict deadline!

Setup Instructions

There are plenty of manuals with lots of information on how to set up Neo4j in your computer. For example, this is the most basic one.

You can already start triggering queries on Neo4j from the browser. In general, you can use this service to query and manage your graph data. Doing so would be the equivalent to use a database-specific IDE (e.g., DBeaver or SQLDeveloper). In the context of this lab, you can use the browser-based service to try and explore, but be aware it does not persist your scripts. Thus, although most probably you want to play and explore with Neo4j from the browser-based service, you must deliver a working program connecting to Neo4j through a driver, as you would do in a real project.

Maybe some other manuals help you better. You are expected to need a couple of hours to set the environment and get familiar with the basics, but this time really depends on each of you.

What to deliver?

A professional way to connect to Neo4j is through a driver from a program in Python, Java or the language you prefer. However, please use Python or Java to create your solutions for this lab. Before developing your solution in another language first contact the lecturer. More information and links to the required drivers for each language can be found here.

You must deliver **an application per exercise**. Please keep the name of the exercise as name of the application plus the surnames of the authors (e.g., *PartA.2_Author1surnameAuthor2surname*).

In addition, you must provide **a document (in PDF)** explaining the required details about each section (more details about the required explanations per section below). The name of this document must be: *DOC_Author1surnameAuthor2surname*. The PDF file must not exceed 8 A4 pages (2.5cm margins, font size 11, inline space 1.15). Documents exceeding this limit might not be considered.

Finally, please upload all the applications together with the lab document as **a single ZIP file** to the corresponding event in Learn-SQL. The name of the ZIP file must be *LAB1_Author1surnameAuthor2surname*.

A Modeling, Loading, Evolving

This exercise is about modeling graph data for a certain domain. The main goal is to develop your skills on modeling and instantiating graph data.

A.1 Modeling

We want to create a graph that stores research publications. In this domain, 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. Each paper can only be published in a single conference, workshop or journal.

A conference/workshop is organized in terms of editions. Each edition of a conference/workshop is held in a given city (venue) and at 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 to a journal, a set of reviewers (typically three) is assigned to each paper. Reviewers are scientists and therefore they are relevant authors (i.e., they have published papers in relevant conferences, workshops or journals). Obviously, the author of a certain paper cannot be reviewer of her own paper.

Note: *The attributes of the concepts are not explicitly given, thus use common sense and the descriptions of the next sections to define attributes for each concept, e.g., a paper contains attributes like Year, Pages, DOI, etc. Be sure to clearly specify the attributes you considered in the lab document.*

Tasks

Perform and explain your solution to the following tasks in the lab document:

1. Create a visual representation of the graph you have designed (in terms of nodes and edges). Clearly specify, within the graph itself or below, the 'schema' of the graph (labels and properties for each node or edge).
2. Justify your design decisions in terms of maintenance, reusability, non-redundancy, ... Besides, consider the performance of queries in Part B in your justification.

Note: *The solution for A.1 must be compiled in the lab document under section A.1.*

A.2 Instantiating/Loading

In this part of the assignment you will have to create a graph following the model designed in the previous section.

You must (partially) instantiate your graph using real data, e.g., load data from Semantic Scholar, DBLP, or any other data source that you may find.

Unfortunately, most data out there is not natively produced as graph data. Therefore, this is a mandatory step in most projects: transforming JSON, XML, or CSV files to graph data. To facilitate such task, most graph databases incorporate built-in bulk load functionalities. For instance, in Neo4j the bulk load is implemented via the LOAD CSV or the Neo4j-admin import commands.

How you implement the loading process is up to you, however you are strongly advised to generate a script to pre-process the real data, so that then you can use a bulk load functionality to load it into the graph. This would be a professional manner to tackle this problem. Furthermore, this pre-processing will also be useful for the Lab on Knowledge Graphs, as the concepts and relationships will be the same and, thus, you will be able to reuse an important part of the work you do now.

Regarding the data sources, some of them such as Semantic Scholar, provide REST APIs to access the data (including the possibility of downloading full datasets in JSON through the API), while others allow you to download the datasets directly from their web. This is the case for DBLP, that makes all its data available in a single XML file. For convenience, we refer you to this tool that converts DBLP data in XML to CSV files compatible with the Neo4j-admin import tool.

Note that your graph does not need to contain all the data available in the sources, but at the same time it should be big enough to allow you to get results for the queries and algorithms in Part B, C, and D. That is, the graph should be useful to test whether your solutions are correct.

Nevertheless, note that the data sources do not contain data for all the concepts and relationships you are asked to create. Thus, you should complete the graph using synthetic data. Just make it reasonable, we are not expecting to see a graph containing only real data. The objective of this section is that you develop good habits when loading data into a graph database.

In any case, we do not recommend to create nodes or edges one by one as it might take quite long!

Note: *The solution for A.2 is a program preparing the data (including both transformations of real data and creation of synthetic data) and loading it into the graph. Besides that, explain in the lab document under section A.2, which data comes from a real dataset (and how you pre-processed it), and which data has been synthetically generated (and how).*

A.3 Evolving the graph

One key aspect of graph databases is their flexibility to absorb changes in the data entering the system. The goal of this part is to experience such nice characteristics.

In the model and instances you created you were asked to identify the reviewers of each paper. Now we want to change this requirement in order to store also the review sent by each reviewer. A review, apart from its content (i.e., a textual description) has a suggested decision. A paper is accepted for publication if a majority of reviewers supported acceptance. Typically, the number of reviewers is 3 but every conference/workshop or journal may have a different policy on the number of reviewers per paper. Furthermore, we also want to extend the model to store the affiliation of the authors. That is, an author is affiliated to an organization which can be a university or company.

Tasks

1. In the lab document, propose a modification of the graph model created in A.1 and highlight the changes introduced. Justify your design for the new/updated elements in the same way as in section A.1.
2. Create an application with the necessary Cypher queries to transform your Neo4j graph (including data and metadata) into an updated graph aligned with the evolved model proposed in the previous item.

Note: *The solution for 1 must be included in the lab document under section A.3, while 2 must be included in a program.*

B Querying

Let us now exploit the graph data. The main goal of this part is to learn how to query graphs. Write efficient Cypher queries (i.e., queries that minimize the number of disk accesses required and the size of intermediate results generated) for the following queries:

1. Find the top 3 most cited papers of each conference/workshop.
2. For each conference/workshop find its community: i.e., those authors that have published papers on that conference/workshop in, at least, 4 different editions.
3. Find the impact factor of the journals in your graph.
4. Find the h-index of the authors in your graph.

Note: *The queries must be included in the lab document under section B and in your program.*

C Recommender

In this task we create a simple recommender. Specifically, we want to create a reviewer recommender for conferences and journals. In this exercise we will identify potential reviewers for the database community.

Notice that this is a process to be performed in several steps (several queries). Importantly, you must assert (i.e., extend the graph to include) the information extracted from each step, and reuse it in the next one. Each step must be implemented by means of a single Cypher query:

1. The first thing to do is to find/define the research communities. A community is defined by a set of keywords. Assume that the database community is defined

through the following keywords: data management, indexing, data modeling, big data, data processing, data storage and data querying.

2. Next, we need to find the conferences, workshops and journals related to the database community (i.e., those that are specific to the field of databases). Assume that if 90% of the papers published in a conference/workshop/journal contain one of the keywords of the database community we consider that conference/workshop/journal as related to that community.
3. Next, we want to identify the top papers of these conferences/workshops/journals. We need to find the 100 papers with the highest number of citations from papers of the database community). As a result we will obtain the top-100 papers of the database community.
4. Finally, an author of any of these top-100 papers is automatically considered a potential good match to review database papers, and we want to include this information in the graph. In addition, we want to identify and store gurus, i.e., reputable authors that would be able to review for top events or journals. Gurus are those authors of, at least, two papers among the top-100 identified.

Tasks

1. For each stage (including the last one), provide a single Cypher statement finding the relevant data and asserting the inferred knowledge into the graph.
2. Justify the design of the new elements in the same way as in A.1.

Note: The required queries must be included in the lab document under section C, together with the justification of the design of the new elements, and in your program.

D Graph algorithms

Let us process graph data with traditional graph metrics / algorithms. To do so, we must use the Neo4j Graph Data Science Library. The library is delivered as a plugin, so you need to install it following the instructions provided in its documentation.

Before going on, verify the plugin has properly been setup in your computer.

Beyond regular queries, graphs allow to exploit graph theory and use well-known graph algorithms. The main goal of this part is to develop your skills on using graph algorithms to query graph data. Graph algorithms are used to compute metrics for graphs, nodes, or relationships and they are parametrized depending on the domain. For instance, the Page Rank algorithm applied to a 'web page search' domain with parameters like ('Page','LINKS'), would find the importance/relevance of the pages taking into account the number and the relevance of links to the page. Similarly, when applying any algorithm you should contextualize it to our domain (i.e., they should make sense in the research publication domain).

The list of algorithms you must study for this lab is the following. Carefully read the algorithms, understand them (with further research if you are not familiar with graph theory), and learn how to parameterize and call them:

1. Centrality algorithms
    - Page Rank
    - Betweenness

- Closeness

2. Community detection algorithms
    - Triangle counting
    - Louvain
    - (Strongly) Connected components
3. Similarity algorithms
    - Node similarity
4. Path finding algorithms
    - Dijkstra's shortest path (from one node to another)

Tasks

- Choose two algorithms from the list above and write calls triggering them correctly (not only syntactically but also semantically).
- For each algorithm used in the previous item, give a rationale of the result obtained from its execution on your graph: i.e., what data are you obtaining? provide an interpretation from the domain point of view. Your answer must show that you understand the graph algorithm chosen and its application on your graph.

Note: *The solution for this section must be included in the lab document under section D and in your program.*

General notes

- One of the criteria for the evaluation of the solutions is the efficiency of queries. You are expected to understand how to design an appropriate graph and how write reasonable queries by understanding the principles on how the graph database works.
- You must create a fair amount of instances. To measure what is fair, you must guarantee that each of the queries and algorithms required in the lab retrieves, at least, an instance of data.
- Use the lab document to justify your solutions and to make any further assumption. Please make sure to include each assumption in the right section.