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. 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

Neo4j is already configured in the lab computers. However, you are recommended to set it up on your laptops too. There are plenty of manuals with lots of information about this, for example, this is the most basic one: `https://neo4j.com/docs/desktop-manual/current/`

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, this 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 (a comprehensive list of the official manuals can be found here: `https://neo4j.com/docs/`). 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 programme 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 link to the required drivers for each language can be found here: `https://neo4j.com/docs/getting-started/current/languages-guides/`.

You must deliver an application, either in Java or Python, per exercise. **Keep the name of the exercise as name of the application plus the surnames of the authors (e.g., `PartA.2_Author1Author2`, `PartA.3_Author1Author2`)**.

In addition, you must generate a lab 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_Author1Author2` (where the suffix are the surnames of the authors). **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_Author1Author2`.

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 modeling 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. 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.

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 to A.1 must be compiled in the lab document under section A.1.

A.2 Instantiating/Loading

1. Define the Cypher expressions to create and instantiate the solution created for the previous section.

Note: The solution for A.2 is a program creating the graph and inserting sample data. Besides that, explain in the lab document, under section A.2, how did you generate this data and any other assumption you want to make explicit about your solution.

You must (partially) instantiate your graph using **real data**, e.g., load the graph from DBLP (`https://dblp.uni-trier.de/`), or from Semantic Scholar (`https://www.semanticscholar.org/`), 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: transform JSON or CSV data to graph data. To facilitate such tasks, most graph databases incorporate built-in bulk load functionalities.

For example, the DBLP data is available in XML format. In Neo4j, the bulk load is implemented via the LOAD CSV command: `https://neo4j.com/docs/cypher-manual/current/clauses/load-csv/`. You can use the following tool to convert DBLP data from XML to CSV: `https://github.com/ThomHurks/dblp-to-csv`, which allows you to generate CSV files that are Neo4j-compatible providing some options, e.g.:

```
python XMLToCSV.py --annotate --neo4j
data/dblp.xml data/dblp.dtd dblp.csv --relations
author:authored_by journal:published_in
```

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

Loading is up to you, however you are strongly advised to generate a small script to pre-process the DBLP and/or Semantic Scholar CSV data, transform it and adapt it

to the graph you designed and use the bulk load functionality to load the graph. This would be a professional manner to tackle this problem. We do **not** recommend to create them one by one as it might take quite long! For bulk loading, you can access a subset of the CSVs (get the piece you want!).
Nevertheless, note the data sources do not contain data for all the graph model 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.

A.3 Evolving the graph

One key aspect of graph databases is their flexibility to absorb changes in the data coming into 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 a certain paper. Now, we want to change such modeling to store 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 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 author. 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 decision.

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.

3. Add the required additional Cypher queries for instantiating the new concepts of your graph in the developed application.

Note: The solution for 1 must be included in the lab document under section A.3, while 2 and 3 in applications created.

B Querying

Let us now exploit the graph data. The main goal of this part is to learn how to query graph data. 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.
2. For each conference find its community: i.e., those authors that have published papers on that conference in, at least, 4 different editions.
3. Find the impact factors of the journals in your graph (see `https://en.wikipedia.org/wiki/Impact_factor`, for the definition of the impact factor).
4. Find the h-indexes of the authors in your graph (see `https://en.wikipedia.org/wiki/H-index`, for a definition of the h-index metric).

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 editors and chairs. In this exercise we will identify potential reviewers for the database community.
Guidelines. 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:

- 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.
- Next, we need to find the conferences and journals related to the database community (i.e., are specific to the field of databases). Assume that if 90% of the papers published in a conference/journal contain one of the keywords of the database community we consider that conference/journal as related to that community.
- Next, we want to identify the top papers of these conferences/journals. We need to find the papers with the highest number of citations from papers of the same community (papers in the conferences/journals of the database community). As a result we would obtain (highlight), say, the top-100 papers of the conferences of the database community.

- Finally, an author of any of these top-100 papers is automatically considered a potential good match to review database papers. In addition, we want to identify gurus, i.e., reputable authors that would be able to review for top conferences. We identify gurus as those authors that are authors of, at least, two papers among the top-100 identified.

Tasks

1. For each stage (including the last one), provide a Cypher statement finding the relevant data of each step and asserting the inferred knowledge into the graph.

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

D Graph algorithms

Let us process graph data with traditional graph metrics / algorithms. To do so, we must use Neo4J's Graph Data Science library `https://neo4j.com/docs/graph-data-science/current/`.

The Graph Data Science library is delivered as a plugin, so you need to install it. Follow the instructions at: `https://neo4j.com/docs/graph-data-science/current/installation/`. You have two options, either through Neo4J's Desktop or by carefully following the steps mentioned in the link above.

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

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 article 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
   - PageRank
   - Betweenness
   - Closeness

2. Community Detection Algorithms
    - Triangle counting
    - Louvain
    - (Strongly) Connected components
3. Similarity Algorithms
    - Node similarity
4. Path Finding Algorithms
    - Dijkstras shortest path (from one node to another)

Tasks

1. Choose two algorithms from the list above and write calls triggering them correctly (not only syntactically but also semantically).
2. For each algorithm used in the previous item, give a rationale of the result obtained: 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 this domain.

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 reaasonable 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 retrieve, at least, an instance of data.
- Use the lab document to make any further assumption of your solutions. Be sure to include each assumption in the right section.