LAB Assignment

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 (look for the corresponding assignment event). 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. The instructions below refer to Neo4j in the lab computers (under the Windows operating system).

1. Locate the Neo4j application under the folder `C:\SOFT`.
2. Run the Neo4j service using: `neo4j console`
3. Open the browser and go to: `localhost:7474`

You can already start triggering queries on Neo4j from the browser-based service at port `7474`. 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).

What to deliver?

To conduct the lab assignment you have two options:

1. You can use the browser-based service. Be aware that you cannot persist your scripts there. You will need to store your scripts aside and structure them **per exercise**. The solution you must hand out would then be this set of scripts.
2. A better alternative (at least, more professional) would be to connect to Neo4j through JDBC-like connectors and create 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/developer/language-guides/`.
   Similarly, create an application, either in Java or Python, **per exercise**.

Summing up, the solution of this lab is either a set of scripts (ready-to-go) or a set of applications (one per exercise in any case). Keep the name of the exercise as name of the script / application plus the surnames of the authors (e.g., `PartA.2_BilalliRomero`, `PartA.3_BilalliRomero`). Aside, you must generate a lab document (in pdf) explaining the required details at each section (more details about the required explanations per section below). The name of this document must be: `DOC_BilalliRomero` (where the suffix are the surnames of the authors). **The PDF file must not exceed 6 A4 pages** (2.5cm margins, font size 11, inline space 1.15). Finally, please upload all the scripts / applications together with the lab document as a single zip file to the corresponding event in Learn-SQL.

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 articles that can be published as scientific papers (papers for short) in the proceedings of a conference/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 proceeding is a published record which includes all the papers presented in the conference/workshop. A conference/workshop is organized in terms of editions. Each edition of a conference is held in a given city (venue) at a specific period of time of a given year. 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 relates to one or more topics through the concept of keywords. Keywords are used by readers to quickly identify the main topics discussed in the paper. To this end, any scientific communication must 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 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., published many 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 ISBN, Year, Pages, DOI, etc.

Tasks

1. Create a visual representation of the graph you would create (in terms of nodes and edges). Use different colours to distinguish data from metadata in your graph.
2. Also, justify your design decisions. Besides maintenance or reusability issues, also consider the performance of queries in Part B when creating your solution.

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 script/program creating the graph and inserting 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.

It is recommended that you use real data, e.g., load the graph from DBLP (`https://dblp.uni-trier.de/`). 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. In our case, DBLP is available in CSV format. In Neo4j, the bulk load is implemented via the LOAD CSV command: `https://neo4j.com/docs/cypher-manual/current/clauses/load-csv/`

Nevertheless, you can create your own instances. In that case, generate enough of them to get results for the queries in Part B, C, and D. Loading is up to you, however you are advised to generate a small script to pre-process some data and use the bulk load functionality, etc. 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 DBLP CSVs (get the piece you want!) from: `https://www.dropbox.com/s/zrfncsu5kpn6zq4/dblp.zip?dl=0` Follow the instructions above to generate the format required by Neo4j to load the data. Please, note that these CSVs do not contain data for all the graph model you are asked to create. They are meant to be used to see an example of how to load external data into a graph database. It is advisable that you use this bulk load functionality to insert data for the rest of the model.

A.3 Evolving the graph

One key aspect of graph data is its flexibility to absorb changes in the data coming into the system. The goal of this part is to experience such nice characteristics.

In the previous model and instances you identified 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. Modify the graph model created in A.1 and highlight the changes introduced. Justify your decision.
2. Write 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. Write additional Cypher queries for instantiating the new concepts of your graph.

Note: The solution for 1 must be included in the lab document under section A.3, while 2 and 3 in the scripts / 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 h-indexes of the authors in your graph (see `https://en.wikipedia.org/wiki/H-index`, for a definition of the h-index metric).
2. Find the top 3 most cited papers of each conference.
3. For each conference find its community: i.e., those authors that have published papers on that conference in, at least, 4 different editions.
4. 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).

Note: The queries must be included in the lab document under section B.

C Graph algorithms

To add the library of the graph algorithms offered by `Neo4j`:
1. Locate the `plugins` folder and download the jar file corresponding to your Neo4j version, e.g., `https://github.com/neo4j-contrib/neo4j-graph-algorithms/releases/download/3.5.4.0/graph-algorithms-algo-3.5.4.0.jar`.
2. Locate the `neo4j.conf` file and add the following line:
`dbms.security.procedures.unrestricted=algo.*`
3. Restart Neo4j.

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 the algorithms you should contextualize them to our domain (i.e., they should make sense in the research article publication domain). Neo4j already provides built-in functions to compute some graph algorithms (see `https://github.com/neo4j-contrib/neo4j-graph-algorithms/`). The algorithms and their input parameters are given below.

- Centrality algorithms
  - `Page Rank (label, relationship, `*`dampingFactor`*`, ...)`
    - `label`: the nodes for which the Page Rank is computed.
    - `relationship`: the relationship (edge) to be taken into account for computing the Page Rank.
    - `dampingFactor`: a value between 0 and 1, used internally by the algorithm to prevent going to an infinite number of 'edges'.
    - Check `https://neo4j.com/docs/graph-algorithms/current/algorithms/page-rank/` for more details.
  - `Betweenness Centrality (label, relationship, `*`direction`*`, ...)`
    - `label`: the nodes for which we want to compute the centrality weight. The centrality weight is a measure of the influence of the node in the graph (or the number of paths for which the node acts as a bridge).
    - `relationship`: the relationship (edge) taken into account when computing the influence of nodes.
    - `direction`: whether the direction of the relationship should be taken into consideration.

    * Check `https://neo4j.com/docs/graph-algorithms/current/algorithms/betweenness-centrality/` for more details.
  - `Closeness Centrality (label, relationship, ...)`
    * `label:` the nodes for which we compute the centrality weight. The centrality weight is a measure of how close are the nodes to all the other nodes (or an average farness of the node to all the other nodes).
    * `relationship:` the relationship (edge) taken into account when computing the closeness.
    * Check `https://neo4j.com/docs/graph-algorithms/current/algorithms/closeness-centrality/` for more details.

- Path finding algorithms
  - `Minimum Weight Spanning Tree (label, relationshipType, startNode, weightProperty, ...)`
    * `label:` the nodes for which we find the subset of the edges connecting all the nodes together, without any cycles and with the minimum possible total edge weight.
    * `relationshipType:` the relationship type that should be explored.
    * `startNode:` the starting point of the algorithm.
    * `weight:` the property name that contains the weight for the edge (relationship).
    * Check `https://neo4j.com/docs/graph-algorithms/current/algorithms/minimum-weight-spanning-tree/` for more details.
  - `Single Source Shortest Path (startNode, weightProperty, ...)`
    * `startNode:` the node for which we want to compute its shortest paths to all the other nodes.
    * `weightProperty:` the property name that contains the weight for the edge (relationship). If not defined the number of hops are used as weights.
    * Check `https://neo4j.com/docs/graph-algorithms/current/algorithms/single-source-shortest-path/` for more details.

- Community detection algorithms
  - `Triangle Counting / Clustering Coefficient (label, relationship, ...)`
    * `label:` nodes for which we want to find if they are part of triangles with respect to a given relationship.
    * `relationship:` the relationship to be exploited when finding the triangles.
    * Check `https://neo4j.com/docs/graph-algorithms/current/algorithms/triangle-counting-clustering-coefficient/` for more details.

- `Louvain (label, relationship, weight, ...)`
  - `label:` nodes for which we want to find the communities to which they belong. The algorithm takes into account the density of communities as compared to how they would be in a random network.
  - `relationship:` the relationship to be exploited when finding the communities.
  - Check `https://neo4j.com/docs/graph-algorithms/current/algorithms/louvain/` for more details.
- `Connected Components (label, relationship, ...)`
  - `label:` nodes in which we want to find the sets of connected nodes (where each node is reachable from any other node in the same set; it only needs a path to exist between pairs of nodes in one direction).
  - `relationship:` the relationship exploited for checking whether a node is connected to another node.
  - Check `https://neo4j.com/docs/graph-algorithms/current/algorithms/connected-components/` for more details.
- `Strongly Connected Components (label, relationship, ...)`
  - `label:` nodes in which we want to find the sets of connected nodes (where connected means that each node is reachable in both directions from any other node in the same set).
  - `relationship:` the relationship exploited for checking whether a node is connected to another node.
  - Check `https://neo4j.com/docs/graph-algorithms/current/algorithms/strongly-connected-components/` for more details.

Tasks

1. Choose two algorithms and write the Cypher queries triggering them correctly.
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 C. Please, note that any of these algorithms might be included in the test conducted to evaluate this lab (regardless of the two you chose to answer this part). Also note that you are expected to research on your own to learn the theoretical foundations (background) of each algorithm.

D 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 page rank provided the number of citations from the 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., very reputated 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, provide a Cypher statement finding the relevant data of each step and asserting the inferred knowledge into the graph.

Note: The solution to this part must be included in the lab document under section D.

General Notes

- One of the criteria for the evaluation of the solutions is the efficiency of queries: both in terms of disk accesses and size of the intermediate results generated by the query.

- 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.