CS 387 Lab 6: Graph Databases

Neo4j Installation:

- Download the community edition from here.
- Unzip the downloaded file
- Run the command : [filename]/bin/neo4j console (neo4j depends on JDK - on different OSs, a different version of JDK. Check the version of JDK you have using java --version and upgrade to the appropriate version based on what neo4j asks for)
- Open the browser and type: http://localhost:7474 (you may have to use Chrome for this).
- The default username and password is: **neo4j/neo4j**

For this lab, you must figure out how to write Cypher scripts to create nodes and relationships to install a graph and then query the graph.

For those unfamiliar with how data is modeled and loaded in neo4j and Cypher queries for querying the database, you can get yourselves familiarized by exploring the inbuilt example datasets that come with the neo4j browser interface.
All you have to do is enter any of the below commands in the Browser command line (at the top of the pane) and follow the accompanying guides to walk through the training or scenarios. These examples are interactive slideshows and can be accessed only with the neo4j browser interface.

- The "Movies" example is launched via the ***:play movie-graph*** command and contains a small graph of movies and people related to those movies as actors, directors, producers, etc.
- The "Northwind" example is run via ***:play northwind-graph*** command and contains a traditional retail-system with products, orders, customers, suppliers, and employees. It walks you through the import of the data and incrementally complex queries using the available data.

There are plenty of other resources available online, feel free to explore. In particular, we recommend THIS one.

Note: For solving the lab you can either use neo4j cypher-shell or browser interface, though browser interface might be slow when creating nodes and relationships to install the graph.

Running the Cypher Shell

- In your installation home directory, run ./bin/cypher-shell
- Use the below command to start using the default *neo4j* database. (Do not use *system* database)

        - :use neo4j

- Let's use Cypher to generate a small social graph.
    - Creating nodes:
        - CREATE (:Person { name: "Emil", from: "Sweden", klout: 99 });
        - CREATE (:Person { name: "Johan", from: "Sweden", learn: "surfing" });
        - CREATE (:Person { name: "Allison", from: "California", hobby: "surfing" });

      CREATE clause to create data
      () parenthesis to indicate a node
      ee: Person a variable 'ee' and label 'Person' for the new node
      {} brackets to add properties to the node

    - Creating Relationships:
        - MATCH (p1: Person{ name: “Emil”}), (p2:Person{ name: “Johan ”}) CREATE (p1)-[:knows]->(p2);
        - MATCH (p1: Person{ name: “Emil”}), (p2:Person{ name: “Allison ”}) CREATE (p1)-[:knows]->(p2);

- Let’s query the graph to find Emil’s friends.
    - MATCH (p1:Person)-[:knows]-(p2:Person) WHERE p1.name = "Emil" RETURN p1, p2;

  The result will be:

```
+-----------------------------------------------------------------------------------------------------------------------+
| p1                                                            | p2                                                    |
+-----------------------------------------------------------------------------------------------------------------------+
| (:Person {name: "Emil", from: "Sweden", klout: 99}) | (:Person {name: "Allison", from: "California", hobby: "surfing"}) |
| (:Person {name: "Emil", from: "Sweden", klout: 99}) | (:Person {name: "Johan", from: "Sweden", learn: "surfing"})       |
```

Lab 6:

For this lab, we are going to use a Synthetic Twitter dataset. There are seven CSV files present in our data. The data model is as depicted below:

Node Types:

- User - properties { name: "" }
- Tweet - properties { text: "" }
- Hashtag - properties { tag: "" }

Relationship Types:

- **Follows** (User-->User)
- **Sent** (User-->Tweet)

- **Mentions** (Tweet-->User)
- **Contains** (Tweet-->Hashtag)

What you must do:

1. With the given set of CSV files that contain data, write a Python program to generate a CYPHER script for creating nodes and relationships in Neo4J. You can execute the file it generates with cypher-shell -f <filename> to load the data.

2. Write CYPHER Queries to answer the following question.

   a. Return those user names in alphabetical order who have mentioned themselves in a tweet sent by them.
   b. Return those tweets that start with ‘we’ and contain the hashtag ‘proud’.
   c. Return those usernames (and the respective tweets) who retweeted tweets of user “Jessica” which do not contain a hashtag.

      Note: If a user X has sent a tweet that is sent by the user “Jessica” then consider it as retweeting.

   d. Return those usernames (and the respective tweets) who retweeted tweets of user “Jessica” which do not contain a hashtag and who are not following user “Jessica”
   e. Return names of top 5 most followed users and the followers count in descending order
   f. Return max number of hops between the user “Ashley” and any other user (Friend of a Friend).
   g. Return names of 5 users in alphabetical order and count, who follow the greatest number of the same users as user “Thomas”, in descending order of the number of same users followed. These users could be thought of as being 'similar' to “Thomas”.
   h. Return all the tweets that mention a user and are sent by someone who doesn’t follow the user.

      Tweets only mentioning the user who sent it should not be included.

      Return user’s name who sent the tweet, user’s name who is mentioned in the tweet, and tweet text.

   i. Return names of all the users who tweet with at least one of the hashtags that “Thomas” also tweets with.
   j. Return up to 2 usernames and the count:

- who are followed by those 5 users who follow the largest number of the same other users as “Thomas” (called similar users).
- Sorted in descending order of occurrences (the number of similar users who follow each).

Submission:

Submit two files:

1. <rollnumber1-rollnumber2>-loaddata.py - this should take the CSV files as input and generate a Cypher script as its output
2. <rollnumber1-rollnumber2>-queries.txt (For each query, identify the ID of the question for which you are submitting your query).

Grading Rubric:

Item	% of grade
Data load script	25
Queries a-f	30
Queries g-j	45

We will run the solution queries we have and match the outputs of your query with the data your cypher script created against what we have.