

Homework Assignment #4

(Neo4J)

Your boss from the MongoDB ZotMusic assignment was once again very satisfied with your work, but went to a party last Friday where there was lots of buzz about graph databases. All of their friends seem to be using them now. You have thus been asked to continue working at the startup, but to use Neo4J for the next set of tasks. In this assignment, you are to get a Neo4J instance working, load a graph version of the ZotMusic data, and run some queries to get some new insights - focusing on connectivity insights - into the ZotMusic data.

Get Ready... (and set)

To start, follow the setup instructions **carefully** in order to install the **correct version** of Neo4J on your machine.

Go...!

It's time to get to work. Here's what you are to do using the Neo4J Browser app inside Neo4J Desktop:

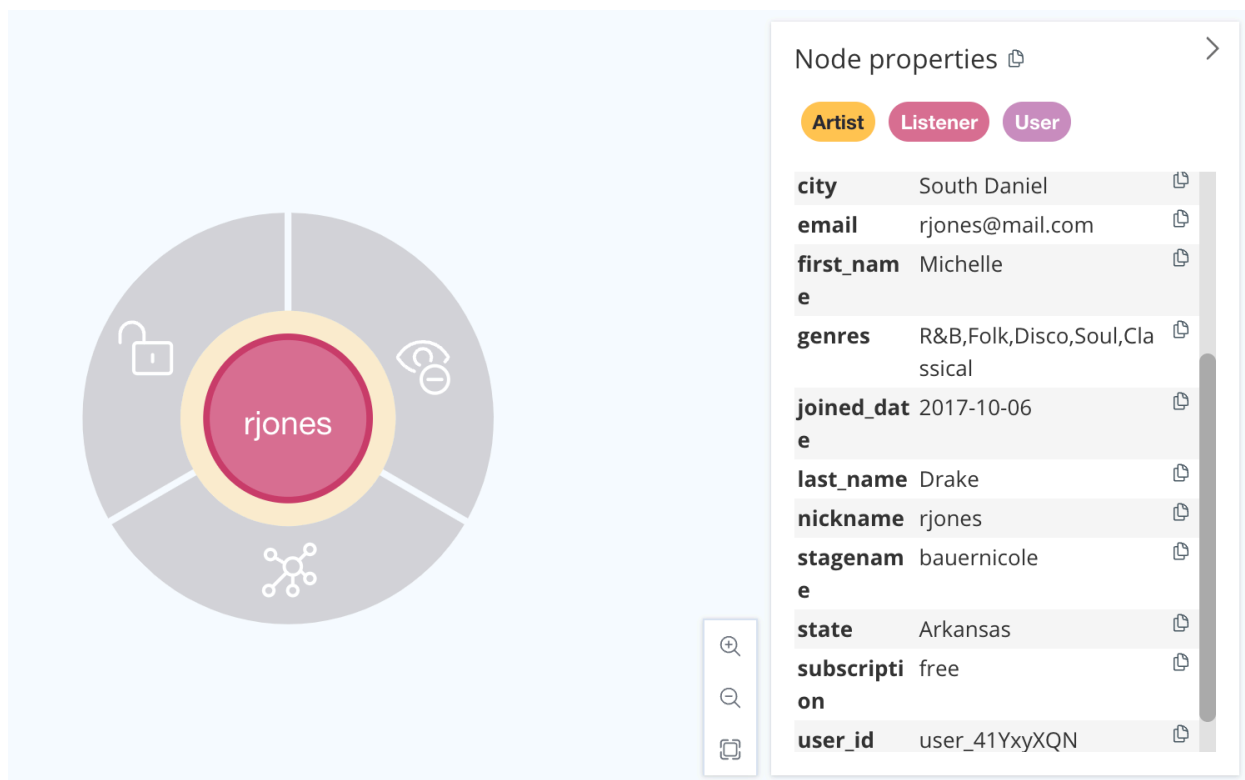
1. Download the HW4 dataset from Canvas. You are given a set of CSV files to load into Neo4J, including the entities (Users, Listeners, Artists, Records, Albums, Singles, Reviews) and the relationships (Artists_Releases_Record, Users_Post_Reviews, Reviews_About_Record, and Upvotes). Note these CSV files reflect a subset of our ZotMusic E-R diagram. (Yes, you no longer need to deal with songs and sessions in HW4!)

You can refer to the Neo4J documentation for commands to import data:

<https://neo4j.com/docs/getting-started/appendix/tutorials/guide-import-relational-and-etl/> .

Note Neo4J does not infer the types of the fields, so you should also include commands to cast them as appropriate types. [Hint: utilize functions like `toInteger`, `toFloat`, etc.]

- A. In your submission, include the commands used to load the data. Remember that some nodes should have multiple labels, for example, a user-labeled node can also have a "listener" and/or "artist" label; similarly, a record-labeled node will have an "album" or "single" label. Ensure that you **merge** nodes when importing data separately for the same node so that each user and record is modeled with just one multi-labeled vertex. For example, below is an image of a node that is labeled with all 3 of the User, Listener, and Artist labels.



Here are the types of possible nodes:

Possible labels for user nodes	Possible labels for item nodes
User, Listener	Record, Album
User, Artist	Record, Single
User, Listener, Artist	

Specifically, in your submission, you should:

- Include commands to load entities.
- To help load the relationship data faster, and also to make MERGEs run faster, you should create indexes on `user_id` and `record_id` for `User` and `Record` respectively. DO THIS IMMEDIATELY after you first load the data from the users and records CSV files (in the sequence of your LOAD ENTITIES steps), as doing it right away will drastically speed up all of the statements that follow. Include the CREATE INDEX statements in your submission.
- Include commands to load relationships.

- B. To show the graph you have just created by loading the data, run the commands below and show the output.

- i. `CALL db.relationshipTypes;`
- ii. `CALL db.labels;`

2. Having a more solid grasp of the nodes and relationships that exist in your graph, you now want to see how the two are intertwined (that is, which relationships are connecting which nodes). Luckily, Neo4J Desktop provides a great visual method for doing this: run the following query to get all nodes with the User label:

```
MATCH (u: User)
RETURN u
LIMIT 10;
```

From the result pane, click one of the nodes and press “Expand child relationships” (the button immediately below your node that looks like a tiny graph). Things to notice (no need to put them in your submission): (i) What types of relationships does a User node have? (ii) What are the labels of the other nodes that these relationships connect to? (iii) Finally, what are the directions of each relationship?

Similar to other “schema-less” systems, two Neo4J nodes with the same label may have different properties, and two relationships of the same type may each have different properties. Nodes aren’t even required to have labels! In Mongo Compass, a schema-analysis tool was offered in their UI in order to sample the documents and return the fields that a document in that collection might have. Neo4J desktop doesn’t offer as straightforward an approach to see what properties a given kind of node might have (though the “Database Information” panel does offer information about the DB as a whole, which you can access by clicking the disk icon in the top left corner.) Nevertheless, you can handle that by issuing a query that will search through a subset of your nodes and extract their property keys. An example for User nodes can be found below:

```
MATCH (u: User)
WITH u LIMIT 1000
UNWIND (keys(u)) AS userKeys
RETURN DISTINCT userKeys
```

Now for the actual question itself: Modify the query immediately above and sample 1000 **Record** nodes for their distinct properties.

3. You are finally ready to write some queries! While writing these Cypher queries, it may help to think about how you would formulate the equivalent SQL statement(s). For the

output/results, please use the *Text* type of output which is shown along with the *Graph* and *Table* formats. Answer each of the questions below:

- A. To start your Cypher query writing journey, your boss has given you the following task: Write a Cypher query to return some User nodes (all properties). Sort the nodes in descending order of `joined_date`, and print just the first 5 nodes.
- B. Cypher (and other graph query languages) must not only be able to search nodes based on their labels, but also be based on the nodes that they are connected to. Write a Cypher query to find nodes with paths from the User node with id `"user_WKWGx1Za"` via the `Releases` relationship. In other words, find the records released by that user. Return the nodes and `release_date` ordered by `record_id` in ascending order.
- C. You might have noticed that some listeners have not reviewed any records. With your curiosity piqued, you now want to find out how many such listeners exist in your graph. Write a Cypher query to count all such listeners.
- D. Write a Cypher query to examine the listeners who have given the record titled "Audience star apply" a rating of 5. Print these listeners' first and last names sorted by their first name in ascending order. Make sure you print distinct listener names.
- E. Write a Cypher query that returns the records that have more than 25 posted reviews associated with them. Print the ids and titles along with the counts of reviews for these records. Sort the results in descending order of counts.
- F. You are interested in knowing which listeners have been relatively active on the platform. Such users will have posted more than 60 reviews and will have also given more than 500 upvotes. Write a Cypher query that returns the ids of active listeners. (*Hint*: You can use a `WITH` clause and aggregations). Sort your query's results in ascending order of `user_id` and return the first 5 results.
- G. The future of ZotMusic depends on increasing engagement on the platform, and your boss wants to increase user engagement by introducing a function on the website where you can meet and talk to listeners with similar tastes for music. To test the desired friend recommendation notion, you would like to start by finding pairs of listeners who have reviewed the same record AND given the same ratings to this record. Write a Cypher query to return the distinct pairs of such listeners along with the `record_ids` of those records. You should follow the following requirements:
 - a. Each pair of nodes should be printed only once (i.e., show either (A, B) or (B, A) but not both!).
 - b. Print only the last names of both listeners and the `record_ids` of those records, and sort by ascending order of the `record_id` and keep only the first 10 results.

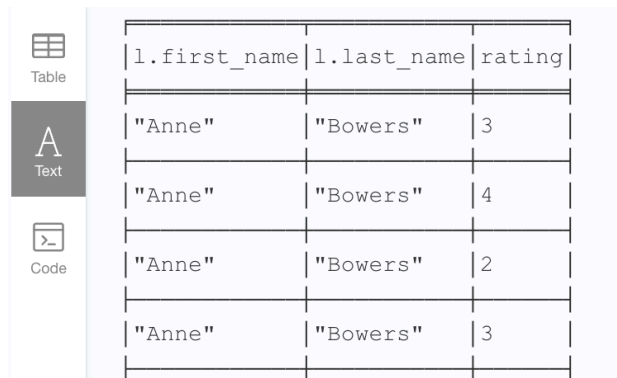
Hint: Every node has a unique id that can be accessed via a function `id(n)` where `n` is the node. These ids can be compared in the usual ways (`<`, `<=`, `=`, `!=`, `>=`, `>`).

- H. Write a Cypher query to return all pairs of users, referred to as A and B, where A has reviewed a record released by B and B has also reviewed a record released by A. Again, (A, B) and (B, A) are considered to be logical duplicates, so only one of the two pairs should be shown by your query. When printing a user, print only the nickname. Sort your results by the nickname of the first user of a pair, and limit your answer to 10 pairs.
- I. You have just received a new request from one of the artists. There is a paranoid artist with user_id "user_0ZIEALBX" who is curious about the potential dissemination of their information to other artists and how quickly it can disseminate. In particular:
- This artist is interested in finding the length of the shortest path from themselves to any other artist. In other words, this artist is wanting to know the length of the shortest path(s) from themselves to any other artist via any nodes and relationships and then getting the minimum length among all those paths. Write a Cypher query that finds the minimum shortest path length from the artist with id "user_0ZIEALBX" to any other artist. (**Hint:** Read up on and take advantage of Neo4J's shortestPath function!)
 - This same artist is now interested in knowing the actual artists(s) involved in the shortest path(s). Write another Cypher query to produce a list of the ids of the other artists(s) whose shortest path is exactly 6 hops (i.e., 7 total nodes in the path) away from the artist with id "user_0ZIEALBX". Sort the results by user_id in ascending order and limit the result set to 5.

What To Turn In

When you have finished your assignment, you should use Gradescope to turn in a PDF file that lists all of the queries you ran -- from start to finish -- and all of the query results as they appear in Neo4j Browser's "Text" format. Follow the steps below in order to generate your PDF file for submission:

- Download the "Homework #4 template.docx" from Canvas.
- Open the file. At the very top, specify your name and student ID.
- Fill in the Cypher query and query results for each question. To fill in the results, use the "Text" format in the Neo4j Browser. You should see a format similar to:



l.first_name	l.last_name	rating
"Anne"	"Bowers"	3
"Anne"	"Bowers"	4
"Anne"	"Bowers"	2
"Anne"	"Bowers"	3

Take a screenshot of the table itself, including all of the borders, for each result that you show. Insert the result screenshots into the template file following each query.

4. Export the template as a PDF file.
5. Finally, submit the PDF file to Gradescope as usual!