CIS 5500: Database and Information Systems

Homework 6: NoSQL

This homework is based on material in Modules 10 and 11. Responses should be submitted via Gradescope using the template files (*hw6_mongo.js*, *hw6_neo.js*). More detailed submission instructions and autograder specifications can be found below in the Submission and Autograder sections respectively.

Overview

In Part 1 (Q1 - Q6, 50 points) you will query a Nobel Prize dataset using Compass. To do this, you should install both MongoDB Community Server and Compass as instructed in the MongoDB handout.

In Part 2 (Q7 - Q12, 50 points) you will use the Neo4j Aura platform to query a dataset on flights and cities.

Advice to Students

This homework assignment is very similar to HW1 in that you should expect to spend some additional time setting up, troubleshooting, and debugging errors with the IDE and other cloud resources. Again, use internet resources such as the official documentation or Stack Overflow to solve setup or other issues. If these resources do not solve your problem, you can ask the course staff for assistance via Ed Discussion or by coming to OH.

You will also notice that this assignment is shorter than HW1. Given that you are already familiar with other cloud resources and the IDE, we do not anticipate this assignment taking any more time than HW1. However, you will need to adapt to a slightly different thought process in writing NoSQL queries compared to the ones you wrote in SQL. Specifically, the aggregation pipeline in Mongodb and graph relations in Cypher are concepts that might take some time to get used to, so plan on spending some time thinking about these.

Again, the staff is happy to help you think through how to approach the query, but will not confirm the correctness of your answers. We do advise you to ***please start early***.

Part 1: MongoDB

(Q1-Q6, 50 points)

Download **prizes.json** and **laureates.json**. This is the collection you will use for the queries.

Follow the instructions in the **MongoDB Handout** to create a local database using **Compass**. Open a console or shell window on Compass to query this database. Upload **prizes.json** and **laureates.json** to the collections characters and locations. You will find Section 3a or 3b of the MongoDB Handout useful here.

Check: In the Mongo shell, make sure you are using the correct database ('use <db-name>' where <db-name> is the name of the database in which the collections are created). Verify that you have done so by running the commands: **db.prizes.countDocuments()**, which should return 585, and **db.laureates.countDocuments()**, which should return 922.

Include just the queries in your submission, not the data. Please make sure that your query output complies with the provided schema where applicable. As you develop the queries, you may want to use pretty() to view the formatted results but it is not necessary in the final answer.

In your queries, use the aggregation framework rather than Map-Reduce (which has been deprecated).

In the template file **hw6_mongo.js**, you will find a set of variables (answer_1, answer_2, etc.). You should store the query that answers the question in these variables. The query should **not** be in the form of a string (paste them into the template file just as you input them into the Mongo shell).

prizes.json contains information about Nobel Prizes, and **laureates.json** contains information about the Nobel Prize laureates.

Question 1. (8 points)

Print the number of Nobel laureates who were born in Germany, in either Berlin or Munich. Your result should be an integer.

Schema: x, x ∈ int

Question 2. (8 points)

Print the year and details of laureates (their first name and surname) who won in the "Peace" category after (but not including) the year 2010.

Schema: {year, laureates (array of documents with firstname and surname)}

As an example, the following should be in the query result:

```
{ year: '2017',  laureates: [{ firstname: 'International Campaign to Abolish Nuclear Weapons
        (ICAN)', surname: '' } ] }
{ year: '2016', laureates: [{ firstname: 'Juan Manuel', surname:  'Santos' } ] }
```

Question 3. (8 points)

Print the first name, surname, and country of all Nobel laureates who were born and died in the same country. Count only those entries where the birth country and the death country are identical. Laureates whose birth and death countries are different due to historical changes should not be included in the count. Your result should be an integer.

Hint: Read about how to use comparison operations within $expr in MongoDB's documentation.

Schema: {firstname, surname, bornCountry}

Question 4. (8 points)

For each affiliation (identified by name in laureates.affiliations), print the number of laureates. Sort the result by affiliation name.

Hint: You can rename the _id field in a result to X using X: "$id"

Schema: {affiliationName, numberOfLaureates}

Your result will include:

```
 { "affiliationName": [], "numberOfLaureates": 254 },
 { "affiliationName": ["A.F. Ioffe Physico-Technical Institute"], "numberOfLaureates": 1 }
```

Question 5. (9 points)

For each year, provide a count of categories where the Nobel Prize in that category was not won by a single individual, with your output sorted by number of categories (descending). In other words, list each year along with how many categories that year were won by more than one individual. Please make sure your answer conforms to the schema and is sorted.

Hint: Read about $cond in MongoDB’s documentation.

Schema: {year, categoriesWithManyLaureates}

Question 6. (9 points)

For each country appearing as a bornCountry in laureates that is not null and has not been renamed (these are the ones with “(now” in the name), print a list of laureates who were born in that country. Include their surname and firstname. Sort the result by bornCountry.

Hint: Read MongoDB’s documentation on regex and $push

Schema: {bornCountry, laureates}

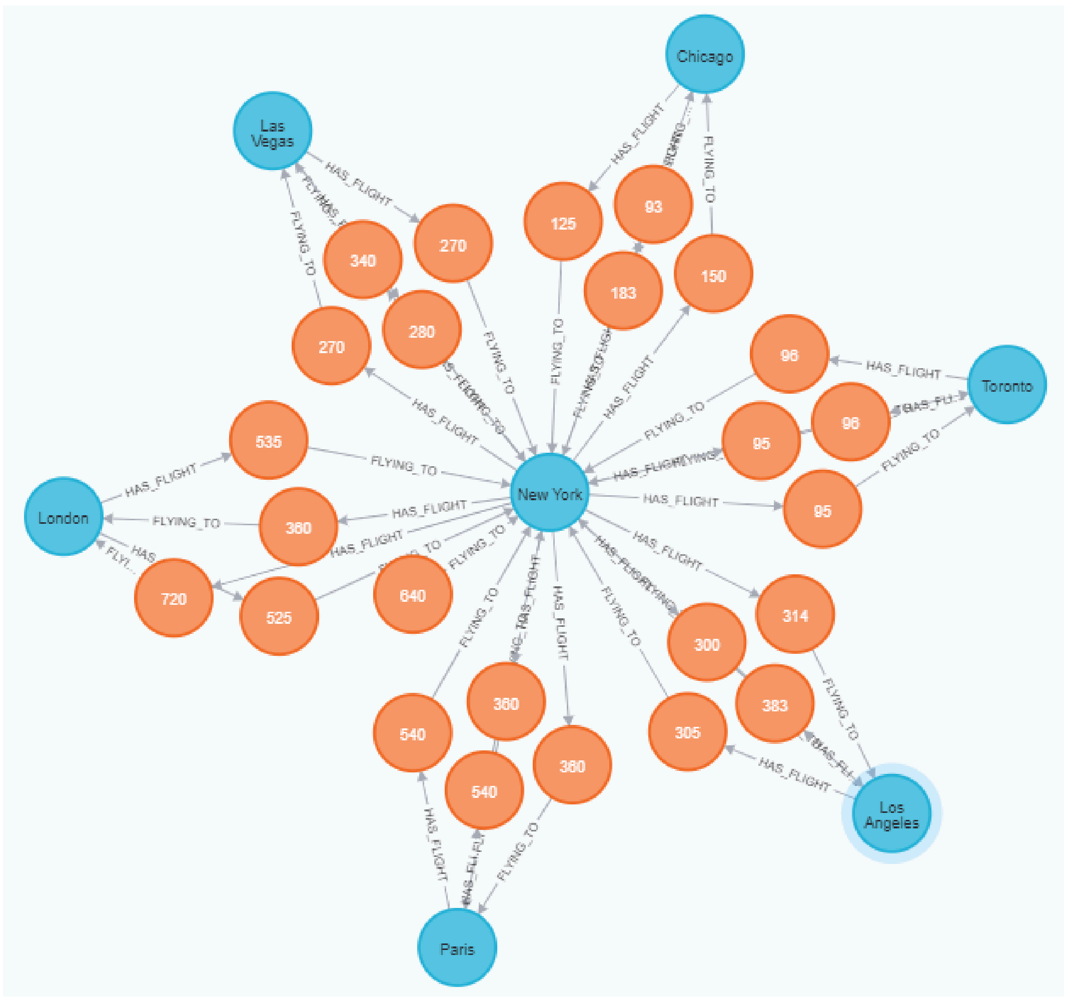
Your answer should include:

```
 { "_id": "Argentina",
   "laureates": [ { "firstname": "Bernardo Alberto", "surname": "Houssay" },
     { "firstname": "César", "surname": "Milstein" },
     { "firstname": "Carlos", "surname": "Saavedra Lamas" },
     { "firstname": "Adolfo", "surname": "Pérez Esquivel" }  ]
 }
```

Part 2: Neo4j

(Q7-Q12, 50 points)

Create a new database on Neo4j Aura and open a new Neo4j browser console as instructed in the Neo4j Handout. In the console make sure to change the database to the one created. Take a look at the data.cyp file, and notice that it is divided into "chunks" separated by line breaks. Paste one "chunk" at a time of data.cyp into the console, hitting the button after each chunk to run. If you try to paste the entire data.cyp in to the console you may get some strange errors – this is a new, undocumented issue!



The figure above is a (sub)graph of the data using the browser console's built-in interactive visualization feature. You will notice that there are two types of nodes here. Flights (marked in orange) have the attributes code, carrier, duration, source_airport_code, departure time, destination_airport_code, and arrival time. Cities have the attributes name and country. There are also two types of relationships - HAS_FLIGHT and FLYING_TO. Two cities c1 and c2 are connected by a flight f using two edges: **(c1)-[:HAS_FLIGHT]->(f)-[|FLYING_TO]->(c2).**

In the template file **hw6_neo.js,** you will find a set of variables (answer_8, answer_9, etc.). You should store the **query** that answers the question in these variables. The query should be in the form of a string enclosed within the provided quotation marks.

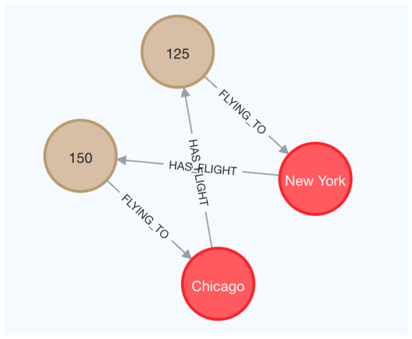
Question 7. (8 points)

Retrieve the flight code, source city name, and duration of the three shortest flights taking off from **either** Australia **or** India, and arriving in Singapore. Note that there may be many flights with the same duration but that your result should only return 3 of the shortest.

Schema: (code, source, duration)

Question 8. (8 points)

Print the subgraph of all flights with a duration strictly less than 4 hours and using United as a carrier. Note that the duration of a flight is expressed in minutes in the dataset. Recall that a subgraph consists of a set of paths, so your query answer should include the source and destination cities (nodes) along with the corresponding flights (edges), as shown below.



Question 9. (8 points)

Retrieve cities with fewer than 5 outgoing flights. Display the count of these outgoing flights (flightCount) and a list of distinct city names that these flights go to (destinations) for each such source.

Schema: (destinations, flightCount, sources)

Question 10. (9 points)

Find all cities that **cannot** be reached from Rome using one flight, but can be using two connecting flights. The first flight's duration must be shorter than 2.5 hours, and the duration of the second flight must be shorter than 3 hours.

In your output, return the intermediate cities and the destination city for each route (intermediate_city, destination), the duration of the first flight, and the duration of the second flight.

Do not include Rome as a destination and ensure that there are no duplicate result tuples.

Schema: (intermediate_city, destination, first_duration, second_duration).

Question 11. (9 points)

Find the five International destination cities with the lowest average duration of day-time direct flights from U.S. cities. Day-time flights are flights that depart at or after 8:00 AM and arrive strictly before 5:00 PM (both of these conditions must be met). The arrival and departure times are the number of minutes from 12:00 AM, and the U.S. is "United States of America".

Schema: {name, length}

Question 12. (8 points)

Suppose we are going on a trip and we're leaving from New York. There are several cities we have in mind as our destinations: London, Athens, and Madrid.

Find the length of the shortest path between New York and each of these 3 cities in terms of the number of flights. Your output should return the name of each of these three cities and the length of the corresponding shortest path from New York.

Schema: {name, length}

Submission

Please submit your responses using the template files (*hw6_mongo.js*, *hw6_neo.js*). **Do not** rename these files. You can add comments to the files, but refrain from changing the order of answers (or variables), or renaming any variables corresponding to answers. Please note that **unlike Mongo queries, Neo4j queries should be submitted as strings**. This is reflected in the template file where Mongo answers default to null and Neo4j answers default to the empty string.

Autograder

You may submit your responses multiple times. Upon submission, the autograder will confirm if your queries were compiled successfully. It will **not** tell you whether or not the query itself is correct.

Please note that queries that do not execute successfully will be heavily penalized.

Manual grading: Upon final submission, your responses will be partially autograded; you will not see these results. We will then do a round of manual checking to adjust the autograded score to adjust (usually, add) credit based on the quality of the responses submitted.