

CIS 550: Database and Information Systems

Homework 5: NoSQL

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

Overview

This assignment has two parts.

Have Compass set up and a cluster ready and connected to MongoDB [Atlas](#) before starting Part 1, as instructed in the MongoDB Atlas Handout.

In Part 1 (Q1 - Q7, 60 points) you will query the Nobel prize dataset hosted on a MongoDB Atlas cluster using DataGrip.

In Part 2 (Q9 - Q12, 40 points) you will use the [Neo4j Aura](#) platform to query a dataset on flights and cites.

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 Piazza 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, map-reduce in Mongo 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-Q7. 60 points)

Follow the instructions in the **MongoDB Atlas Handout** to create a cluster with your MongoDB Atlas account, and open a new console window in Compass, connecting to it.

Download prizes.json and laureates.json and upload them to the collections prizes and laureates respectively. You will find Section 5 of the **MongoDB Atlas 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.count** = 585, and **db.laureates.count** = 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 the template file answers.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. As you answer these questions, you might come up with single queries that answer the question, or a set of statements that define helper variables and functions. You will need to include the definitions of these helper variables and functions in your submission file. The following examples show how you can properly place your queries in the submission file to ensure that they are graded successfully.

If your answer to question 2 is a single query that looks similar to the following

```
db.prizes.count();
```

You should have the following code in your submission file

```
var answer_2 = db.prizes.count();
```

If your answer to question 3 is a set of statements that define variables or functions used in your final query, such as the following

```
var mapFunction = function() {  
    emit(this.id, this.amount);
```

```
};
```

```
var reduceFunction = function(key, values) {  
    return Array.sum(values);
```

```
};
```

```
db.orders.mapReduce(mapFunction, reduceFunction,  
    {out: {inline: 1}});
```

You should have the following code in your submission file (note the bolded lines). In this example, note that the bolded lines come after the definitions of the 'mapFunction' and 'reduceFunction'.

```
var mapFunction = function() {
    emit(this.id, this.amount);
};
```

```
var reduceFunction = function(key, values) {
    return Array.sum(values);
};
```

```
var answer_3 = db.orders.mapReduce(mapFunction, reduceFunction, {out:
{inline: 1}});
```

Question 1. (7 points)

Print the number of Nobel Laureates who were either born in "Philadelphia, PA" or affiliated with "University of Pennsylvania". Your result should be an integer. (Go Quakers!)

Question 2. (7 points)

Print the first and last names of all male Nobel Prize Laureates who have won a Nobel Prize in either peace or economics.

Schema: {firstname, surname}

Question 3. (9 points)

For each affiliation, print the number of Laureates associated with that affiliation. Return the result sorted in decreasing order of the number of Laureates.

Schema: {_id, num}}, where the value of _id is the affiliation name (recall the aggregation pipeline).

Question 4. (9 points)

Print the birth country with the maximum number of Nobel Prize Laureates born after the year 1900 who have won a Nobel Prize for a discovery, that is, the motivation field contains the word 'discovery'. If there are multiple such countries, just return one. Also include the number of such laureates in the result.

Schema: {_id, number}, where the value of _id is the birthCountry (recall the aggregation pipeline).

Hint: Note that the birth date is a string, not a date. You will also need to use the find command with a regex - see [here](#).

Question 5. (9 points)

Using prizes, print a relational (non-nested) version of the dataset for prizes in Physics after the year 2015. Retain only the keys year, id, firstname and surname. For example, if the only matching document in prizes.json were the first document the result would be:

```
[{ "year": "2017",
  "id": "941",
  "firstname": "Rainer",
  "surname": "Weiss" },
{ "year": "2017",
  "id": "942",
  "firstname": "Barry C.",
  "surname": "Barish" },
{ "year": "2017",
  "id": "943",
  "firstname": "Kip S.",
  "surname": "Thorne" }
]
```

Schema: [year, id, firstname, surname]]

Question 6. (10 points)

For each year, print the total number of Nobel Prizes awarded over all categories. Note that where one prize is awarded to 3 people, we should count it as 3 prizes awarded. Your output should contain a results array of objects with keys (_id, value), where '_id' is the year and 'value' is the total count. You must use MapReduce for this part.

Schema: [{_id, value}]


Question 7. (9 points)

While MapReduce is a fundamental programming model to learn, MapReduce queries in MongoDB can be written as aggregations. MongoDB is more optimized for aggregation pipelines. You will find the official Mongo [documentation](#) helpful in this conversion and in understanding both concepts at a more abstract level.

Rewrite your answer to Question 5a using an aggregation pipeline query.

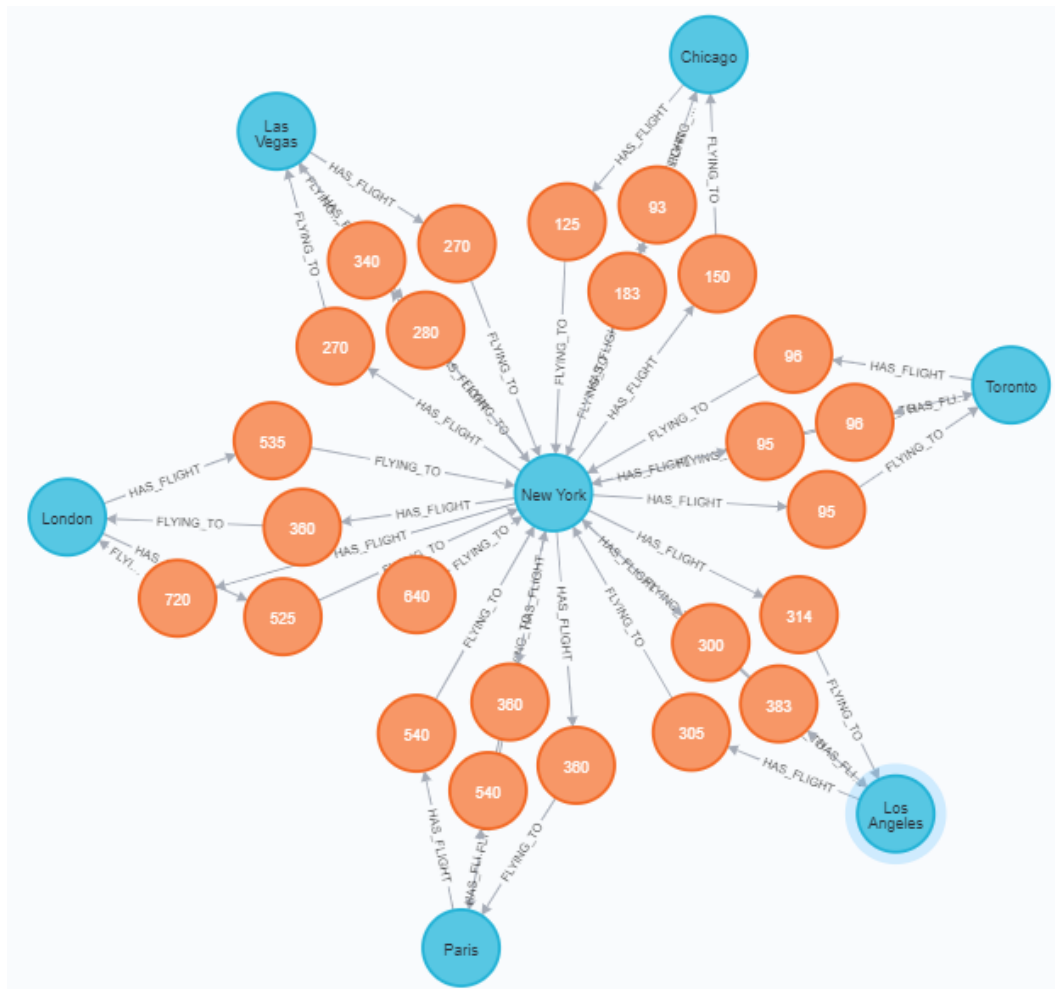
Part 2: Neo4j

(Q8-Q12. 40 points)

Create a new database on Neo4j Aura and open a new Neo4j browser console as instructed in the Neo4j Handout. Paste the contents of [data.cyp](#) into the console and hit the  button to run. Then, execute the following two queries separately in the command line:

```
CREATE CONSTRAINT ON (city:City) ASSERT city.name IS UNIQUE;
```

```
CREATE CONSTRAINT ON (flight:Flight) ASSERT flight.code IS UNIQUE;
```



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 c_1 and c_2 are connected by a flight f using two edges:

$(c_1)-[:HAS_FLIGHT]->(f)-[:FLYING_TO]->(c_2)$.

Question 8. (7 points)

Write a query to add a new city node with value {"name": "Paris", country:"United States of America"}. Notice what happens when you execute it!

Question 9. (7 points)

Print the subgraph of flights whose destination is "London", including sources.

Schema: (source, flight, destination)

Question 10. (8 points)

Return all airport codes that start with "A" or "L". Eliminate duplicates.

Hint: Note that an airport code may appear as a source_airport_code or a destination_airport_code. Also see about [regular expressions](#) in the WHERE clause.

Schema: (airport_code)

Question 11. (9 points)

For each city, print the destinations that can be reached via some flight. Be sure to eliminate duplication destinations. For example, there are 6 flights out of Rome ({"name": "Rome", "country": "Italy"}) but only 3 different destinations that can be reached (New York, Paris and Athens). Order the result by the name of the city.

Schema: (city, destinations[name])

Question 12. (9 points)

Print all cities that can only fly to destinations within the same country. That is, all of the flights out of that city have a destination which is within the same country.

Schema: (name)

Submission

Please submit your responses using the template files (homework5_part1.js and homework5_part2.cyp). **Do not** rename these files. Do not add comments to the files, change the order of answers (or variables), or rename 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 executed successfully and whether the schema is correct; if a query is executed successfully it will also tell you the number of rows in the query result, and a (possibly condensed) preview of the result of executing your query. 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.