CIS 5500: Database and Information Systems

Homework 6: NoSQL and Vectors in Postgres

This homework is based on material in Modules 11 and 12 and is worth 60 points.

Responses should be submitted via Gradescope using the template files: *hw6_mongo.js*, *hw6_neo.js, and hw6_vector.py*. More detailed submission instructions and autograder specifications can be found below in the Submission and Autograder sections respectively.

Overview

In Part 1 (Q1 - Q3, 25 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 (Q4 - Q6, 25 points) you will use the Neo4j Aura platform to query a dataset on flights and cities.

In Part 3 (Q7 - Q8, 10 points) you will use the vector database extension, pgvector, in Postgres to query the IMDB database used in the first part of HW1.

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

(Questions 1-3, 25 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 prizes and laureates. 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 916.

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)

Find all Nobel prize in medicine laureates who won their prize between 1950 and 1970 (inclusive), and who were born in either Germany or the UK. Display their first name, surname, and birth country (aka "bornCountry").

Schema: {firstname, surname, bornCountry}

Question 2. (8 points)

Print the number of Nobel Prize winners who have won more than one Nobel Prize **using the laureates.json dataset**. Name the output column "numWinners".

Schema: {numWinners}

Question 3. (9 points)

Print all the Nobel Prize winners who have won more than one Nobel Prize **using the prizes.json dataset**. List your results in descending order based on first the number of prizes, and then in ascending order by surname. For each winner list their first/last name, number of prizes won, and the categories which they won them for.

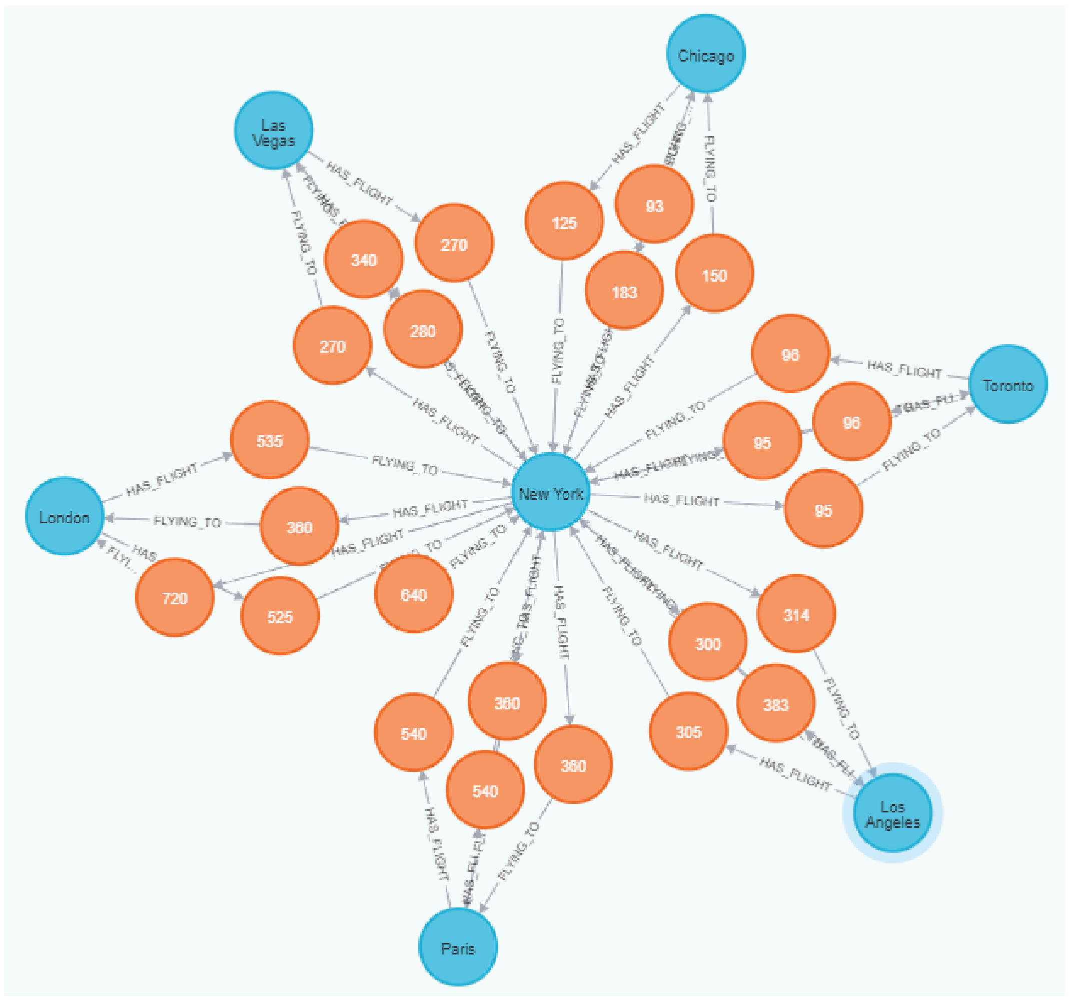
Hint: You may use sort.

Schema: {firstname, surname, numPrizes, categories}

Part 2: Neo4j

(Q4-Q6, 25 points)

Create a new database on Neo4j Aura and open a new Neo4j browser console as instructed in the Neo4j Handout. Follow the directions to load the data from flights.cyp.



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_4 answer_5, 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 4. (7 points)

Find all direct flights operated by carriers whose code begins with the letter ‘D’ and have a duration greater than 2hrs and 45mins. Return the flight’s code (as ‘code’), carrier (as ‘carrier’) and duration (as ‘duration’), sorted in descending order by duration (longest flights first).

Schema: {code, carrier, duration}

Question 5. (9 points)

Print all **paths** of possible single-stop routes from “New York” to “Los Angeles”. The route must have a layover at an intermediate city of between 1 and 10 hours (inclusive). Note: departure and arrival times are in minutes from 12:00AM, and you may ignore layovers that occur overnight. Each path should include source city, destination city, intermediate city and the 2 flights (nodes and edges from the route).

Question 6. (9 points)

You are making a vlog, exploring every city that is reachable from New York that doesn’t have direct flights to it. To build your itinerary, you need to find the shortest number of flights from New York to each of the destination cities. Return each destination city’s name and the shortest path length from New York in ascending order of path length.

Note: Assume a maximum of 6 hops (12 relationship traversals)

Schema: {destination_city, shortest_path_length}

Part 3: VectorDB

(Questions 7-8, 10 points)

For this section, use the IMDB database instance provided in HW1. There is an additional table, synopses, that contains in each row a long text synopsis for a movie and a vector embedding that encodes this text (using ChatGPT's embedding API). To understand the syntax of pgvector (the Postgres extension we use for vector operations), you can refer to its Github page or other online resources.

Note: Distances from pgvector are double precision types and do not play well with the ROUND function. Cast them to numeric before rounding like so: (e1 <=> e2)::numeric.

Question 7. (5 points)

For all movies that have a synopsis, find the 10 movies that are most similar to the movie titled "The Insatiable" which was released after the year 2000. Use cosine distance (<=>) on the synopsis embeddings (synopses.embedding), where a lower distance means more similar. Exclude the target movie itself from the results. Order the output by distance in ascending order, then by title in alphabetical order. Round the distance to 5 decimal places.

Schema: (title, distance)

Question 8. (5 points)

For all movies released after 2010 that have a synopsis, pair each Comedy movie with its single most similar Horror movie using cosine distance on the synopsis embeddings (lower = more similar).
From those per-Comedy nearest matches, return 15 pairs total, ordered by distance ascending, then comedy and horror (alphabetically). Round the distance to 5 decimal places.

Hint: To get exactly one Horror movie per Comedy (the nearest match), you'll need to use PostgreSQL's DISTINCT ON clause (documentation link here), which returns the first row of each group after ordering.

Schema: (comedy, horror, distance)

Submission

Please submit your responses using the template files (hw6_mongo.js, hw6_neo.js and hw6_vector.py). **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 executed 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.