|  |
| --- |
| **CIS 450/550: Database and Information Systems** |
| **Homework 5** |

Due by 23:59:59 EDT on July 1, 2019

\*\*Note: No late submissions will be accepted, so we can release solutions prior to the final\*\*

# Part 1: MongoDB​ (50 points)

For this part of the homework, you will be writing queries in MongoDB. For more information on

the language, please refer to <https://docs.mongodb.org/manual/tutorial/query-documents>.

## Importing​ data​ to MongoDB​ and​ connecting​ to​ the​ database:

This dataset you will use contains information about Nobel Prizes and Laureates. More

**information on the Dataset** can be found at <https://nobelprize.readme.io/docs/getting-started>.

Use the instructions from the in-class exercise to run Mongo and import the dataset (repeated below):

1. Download [**prizes.json**](https://drive.google.com/a/seas.upenn.edu/file/d/1e7ven4pL2NVwINofvemJ2CXB0dZrUPQo/view?usp=sharing) and[**laureates.json**](https://drive.google.com/a/seas.upenn.edu/file/d/1VLuqquKjUFi2YFK0Je3aauk6L1GYIPpZ/view?usp=sharing).
2. *Run the Mongo daemon:*
   1. Mac users and Window users who have mongo path set, open terminal/command prompt and type **mongod**
   2. If mongo path is not set, open the terminal/command prompt, navigate to the folder where MongoDB is installed. Navigate to the bin folder(it would be under MongoDB/Server/3.4) and then type **mongod**
3. *Import the documents:* With the Mongo daemon running in one terminal, open another terminal/command prompt similar to the one in Step 2, type the following:

mongoimport --db nobel --collection prizes --drop --file ~/Downloads/prizes.json --jsonArray

mongoimport --db nobel --collection laureates --drop --file ~/Downloads/laureates.json --jsonArray

(where ~/Downloads/prizes.json is the location of the prizes.json file). You **may need to change this part** to match the appropriate file location on your computer. Make sure you use the same database and collection names!

More information on mongoimport can be found at <https://docs.mongodb.com/manual/reference/program/mongoimport/>

1. *Run the Mongo shell*: With the Mongo daemon running in one terminal, open another terminal/command prompt similar to the one in Step 2, type **mongo.**
2. *Connecting to your database*: Now that you’ve run the mongo shell, type **use​ nobel.** This will shift to using the nobel database (you should see a line “switched to db nobel”).
3. Run **db.prizes.count()** ​and **db.laureates.count()** ​and make sure that they output 585 and 922, respectively.

**To​ stop​ the​ Mongo​ daemon​ and​ shell,​ type​ Ctrl+C.​ To​ start,​ perform​ steps​ 2,​ 4​ and​ 5.**

## MongoDB Questions:

Include just the queries in your submission, not the data. As you develop the queries, you may want to use pretty() to view the results in a formatted way but this is not necessary in the final answer.

1. (5 points): Print the (name of) countries in which Nobel Laureates were born, ordered by country name. Eliminate duplicates.

Schema: (bornCountry)

db.laureates.distinct("bornCountry").sort()

1. (5 points): Print the number of Nobel Prizes in *physics* (a category), assuming that a single prize awarded to several laureates counts as just one. (Result is an integer.)

db.prizes.find({category:"physics"}).count()

1. (7 points):​ Print the name of the category in which the most Nobel prizes have been awarded where, unlike question 2, a prize awarded to 3 people counts as 3 prizes awarded

Schema: (category)

db.prizes.aggregate([

{$project:{category:1,laureates:1}},

{$unwind:"$laureates"},

{$group:{\_id:"$category", count:{$sum:1}}},

{$sort: {count: -1}},

{$limit:1},

{$project:{category:"$\_id", \_id:0}}

])

1. (6 points):​ Print the first name and surname of Nobel Laureates who won a Nobel prize in *literature* and who died in either *France* or the *United Kingdom*.

Schema: (firstname, surname)

db.laureates.aggregate([

{$match:{"prizes.category": "literature", diedCountry:{$in:["France", "United Kingdom"]}}},

{$project:{firstname:1, surname:1, \_id:0}}

])

1. (6 points): ​Print the first name and surname of Laureates who have received more than one Nobel prize.

Schema: (firstname, surname)

db.prizes.aggregate([

{$project:{laureates:1,\_id:0}},

{$unwind:"$laureates"},

{$group:{\_id:"$laureates.id", firstname:{$last: "$laureates.firstname"}, surname:{$last: "$laureates.surname"}, count:{$sum:1}}},

{$match:{count:{$gt:1}}},

{$project:{firstname:1,surname:1,\_id:0}}

])

1. (8 points)

This question uses both the prizes and laureates dataset.  
John Bardeen won the Nobel prize in Physics twice. He shared the prizes with other laureates. For each of those laureates, print their date of birth.

Schema: (born)

var share\_id = db.prizes.aggregate([

{$project:{laureates:1,\_id:0}},

{$match:{ "laureates.firstname": "John", "laureates.surname": "Bardeen"}},

{$unwind:"$laureates"},

{$match:{ "laureates.firstname": {$ne:"John"}, "laureates.surname": {$ne:"Bardeen"}}},

{$project:{"laureates.id":1,\_id:0}}

]).toArray()

var arr =[];

for(var i=0;i<share\_id.length;i++){

var rst=db.laureates.find({id: share\_id[i].laureates.id},{born:1,\_id:0}).toArray();

arr.push(rst);

}

arr

The next two questions use the Map Reduce framework offered by MongoDB (see <http://docs.mongodb.org/manual/core/map-reduce/>). For each question write a single map reduce job to do the following. Do not create/modify a collection to store your map reduce results. You may use out:{inline:1} to display the results on your terminal.

**Note:**  It is difficult to rename keys in the result of mapReduce, so we will be flexible in what the "Schema" of the result is. E.g. for the first question year => \_id and count =>value is ok.

1. (6 points):​ For each year, print the number of Nobel Prizes awarded in any category (where one prize awarded to 3 people counts as 3 prizes awarded).

“Schema”: (year, count).

var mapFunction1 = function () {

for (var idx=0; idx<this.laureates.length; idx++) {

var key= this.year;

var value= 1;

emit(key,value); } }

(it is not efficient for the above query)

var mapFunction = function() {

emit(this.year, this.laureates.length); }

var reduceFunction1= function(key, vals) {

return Array.sum(vals); }

db.prizes.mapReduce(mapFunction1, reduceFunction1, {out:{inline:1}} )

1. (7 points): ​For each country in which Laureates have been born, print the **average** number of Nobel prizes won. Note that we are looking for the average number of prizes, not Laureates.

“Schema”: (Country, Count).

var mapFunction2 = function () {

var key= this. bornCountry;

var value= { count:1, qty: this.prizes.length };

emit(key,value); }

var reduceFunction2= function(key, vals) {

var reducedVal = {count:0, qty:0}

for (var idx=0; idx< vals.length; idx++) {

reducedVal.count += vals[idx].count;

reducedVal.qty += vals[idx].qty; }

return reducedVal; }

var finalizeFunction2= function (key, reducedVal) { reducedVal.avg= reducedVal.qty/reducedVal.count; return reducedVal; };

db.laureates.mapReduce(mapFunction2, reduceFunction2, {out:{inline:1}, finalize: finalizeFunction2 } )

# 

# Part 2: Neo4J (50 points)

## Setting up Neo4J:

You will already have installed Neo4j for the in-class exercises, but instructions can be found [​here​](https://docs.google.com/document/d/1P3PArDfRb_8GK0ZNziTlkuuKGNsHztf0UykajfkRO0k/edit).

Download [data.cyp](https://drive.google.com/a/seas.upenn.edu/file/d/1APvjCOJyxmedbsznVMqjy5Tz3tJ1XRIO/view?usp=sharing). Copy the contents of the file and paste it in your Neo4j terminal. Alternatively, follow the instructions at the above link to import the data file into Neo4j.

## Cypher Queries:

9. (10 points)

Understand the syntax for creating new nodes and relationships from the script in data.cyp and update the database as follows.

1. Create new city nodes for each of the following cities *using a single query instead of using a sequence of create queries*:   
   (**Hint:** use WITH and FOREACH.)

WITH [

{name: "Philadelphia", country:"United States of America"},

{name: "Washington", country:"United States of America"},

{name: "Beijing", country:"China"},

{name: "Shanghai", country:"China"},

{name: "Berlin", country:"Germany"},

{name: "Munich", country:"Germany"}] AS cities

FOREACH (c in cities | CREATE (:City{name:c.name, country:c.country}))

1. Create a new Flight with Flight code=CH007, carrier = China Airlines, duration =360, Source\_airport\_code=SHA, Destination\_airport\_code= SIN, departure=125 and arrival=725.

CREATE(:Flight{code:"CH007", carrier:"China Airlines", duration:360, source\_airport\_code:"SHA", destination\_airport\_code:"SIN", departure:125, arrival:725})

1. Now create the FLYING\_ TO and HAS \_FLIGHT Relationships for the flight above.

MATCH (s1:City {name:"Shanghai"}),(d1:City {name:"Singapore"}),(f1:Flight{code:"CH007",carrier :"China Airlines",duration:360,source\_airport\_code:"SHA",destination\_airport\_code:"SIN",departure:125, arrival:725})

CREATE (s1)-[:HAS\_FLIGHT]->(f1)-[:FLYING\_TO]->(d1)

10. (6 points)

Return the duration of all flights whose carrier is “United” and sort the result in ascending order. Your output should have schema (time​).

MATCH(f:Flight{carrier:"United"})

RETURN f.duration AS time

ORDER BY f.duration

11. (6 points)

Return all carriers that have a flight to JFK. Eliminate duplicates.

MATCH(f:Flight{destination\_airport\_code:"JFK"})

RETURN DISTINCT f.carrier

12. (6 points)

​Return the subgraph of flights out of “Las Vegas”, including the destinations. (Have some fun with this output, try dragging the nodes around and see how the graph rearranges!)

MATCH p=((:City {name:"Las Vegas"})-[:HAS\_FLIGHT]->(f:Flight)-[:FLYING\_TO]->(:City))

RETURN p

13. (7 points)

Return the names of all cities that can be reached from “New York” via exactly one intermediary airport. Sort those names alphabetically and remove any duplicates in the output. Your output should have schema (name​).

MATCH (:City{name:"New York"})-[:HAS\_FLIGHT]->(:Flight)-[:FLYING\_TO]->(:City)-[:HAS\_FLIGHT]->(:Flight)-[:FLYING\_TO]->(d:City) RETURN DISTINCT d.name AS name ORDER BY d.name

14. (7 points)

Return a list of all cities and their airport codes. Again, be careful to include source as well as destination airport codes, and eliminate duplicates.

Your output should have schema (name, code).

MATCH (c1:City)-[:HAS\_FLIGHT]->(f1:Flight)

RETURN DISTINCT c1.name AS name, f1. source\_airport\_code AS code

UNION

MATCH (f2:Flight)-[:FLYING\_TO]->(c2:City)

RETURN DISTINCT c2.name AS name, f2. destination\_airport\_code AS code

15. (8 points)

For each country with more than one city, return the number of cities. Your output should have schema (country​, num​).

MATCH (c: City)

WITH c.country AS country, count(\*) AS num

WHERE num>1

RETURN country, num