CSPB 3287: Lab #6

MongoDB

Overview

This assignment will give you hands-on practice in working with a NoSQL database designed for handling Big Data. A MongoDB server with a database is provided and you will analyze the data to provide answers to questions.

This assignment will allow you to learn how to search the NoSQL database and use commands to group, aggregate, and sort results.

Objectives

- Analyze data stored in a MongoDB database.
- Become familiar with syntax and commands of MongoDB interface.

Submission Requirements

- Compose answers to the questions assigned below in a document. Answers must be clearly identified and numbered according to the questions below.
- Where the question requires a MongoDB command, submit both your command code and your answer set. Some questions might produce a lot of output. Please only copy/paste approximately the first 20 lines of output into your final submission document.
- Save the document as a PDF and submit on Moodle.

Step 1: Connecting to MongoDB Server

This assignment will require you to connect to the MongoDB Server through the virtual machine for 3270 from coding.csel.io. However, you will be using a command line interface to the MongoDB. This will allow you to use the ***mongosh*** command to create a terminal connection for a command line interface.

You will use the following credentials to access the MongoDB Server.

```
$ mongosh "mongodb://student:CspbStudent1234@applied-
sql.cs.colorado.edu/HW_6_db"
```

Once you have established a connection, you will see a command prompt with the name of the assignment database. If you have a different prompt, issue the ***use HW_6_db*** command to switch to the assignment database.

```
other-db> use HW_6_db
switched to db HW_6_db
HW_6_db>
```

Each Mongo database can have multiple collections. List out the set of collections available in the database. For this assignment, only the ***HW_6_data*** collection should be listed.

```
HW_6_db> show collections
HW_6_data
```

You should "play" with the data before trying to complete the assignment. Below is a set of example queries and the resulting output.

List out the data by issuing a query to the database. You will use the ***find*** command to collect a set of information. Each command will specify that you are using the current database, the collection within the database, and the command to use to process the data. Remember that you will have compound commands where one function will process the data and pass it to a second command. For example, in the code below the find() function is passing information to the pretty() function. The pretty printer will format the JAON information to make it more readable.

```
HW_6_db>  db.HW_6_data.find().pretty()
[
  {
    _id: ObjectId('669c50612a41d8b4923c7786'),
    address: {
      building: '1007',
      coord: [ -73.856077, 40.848447 ],
      street: 'Morris Park Ave',
      zipcode: '10462'
    },
    borough: 'Bronx',
    cuisine: 'Bakery',
    grades: [
      {
        date: ISODate('2014-03-03T00:00:00.000Z'),
        grade: 'A',
        score: 2
. . .
]
Type "it" for more
HW_6_db>
```

There is too much data to be displayed and the server will limit the amount of data returned. If you want to see more than the 25 rows, you can use the ***it*** command to get the next iteration of rows. How can you count the number of rows available in the database collection? Using the ***find()*** command without any parameters will get the whole collection. You can pass those results to a ***count()*** function to print the number of result rows.

```
HW_6_db>  db.HW_6_data.find().count()
25359
```

Let's find the number of restaurants where the name starts with the word 'The' and then print out the count. Creating queries is difficult and should be taken one step at a time. First list out the data for the first 5 matches and only print out the names. Once you know you are only getting the data you wanted, expand that working query to continue the processing. In this case, now calculate the count.

```
HW_6_db>  db.HW_6_data.find({"name":/The/}).pretty()
. . . Lots of data . . .
HW_6_db>  db.HW_6_data.find({"name":/The/}).limit(5)
. . . Lots of data . . .
HW_6_db> db.HW_6_data.find({"name":/The/}, {name:1, _id:0}).limit(5)
[
  { name: 'Brunos On The Boulevard' },
  { name: 'Taste The Tropics Ice Cream' },
  { name: 'The Movable Feast' },
  { name: 'The Country Cafe' },
  { name: 'The New Starling Athletic Club Of The Bronx' }
]
```

You will notice there are matches that do not begin with the correct word. The ***find()*** function will pattern-match similar to the regular expressions used by the UNIX ***grep*** command. You can use the `^` character to specify the beginning of a string.

```
HW_6_db> db.HW_6_data.find({"name":/^The/}, {name:1, _id:0}).limit(5)
[
  { name: 'The Movable Feast' },
  { name: 'The Country Cafe' },
  { name: 'The New Starling Athletic Club Of The Bronx' },
  { name: 'The Princeton Club' },
  { name: 'The Punch Bowl' }
]
```

Now that you know the matching is performing as you would like it, change the ***limit()*** to a ***count()*** to count all the data found to match.

```
HW_6_db> db.HW_6_data.find({"name":/^The/}, {name:1, _id:0}).count(5)
782
```

There are sections of New York City and each has a unique name. We can query the data to get a list of those names. Collect the ***borough*** data from all the rows, but only print out each borough once. Use the ***distinct()*** command.

```
HW_6_db> db.HW_6_data.distinct("borough")
[
  'Bronx',
  'Brooklyn',
  'Manhattan',
  'Missing',
  'Queens',
  'Staten Island'
]
```

For a last example of coding a query, we will collect information from the data and aggregate it. How many Greek restaurants in each of the boroughs? Using the ***aggregate()*** function, collect the data grouped by ***borough*** and matching Greek ***cuisine***, and then count the number of items in each ***borough***. You will want to create this query in steps to make sure you are counting the correct data. Try getting the data from one borough and looking to see you have the correct data using the ***find()*** function. Check the count of the results. Change to list out the data from a different borough. When it works for any borough, try the aggregation.

```
HW_6_db> db.HW_6_data.aggregate([
                                {$match: {cuisine : "Greek"}},
                                {$group: {_id: "$borough", count: {$sum: 1}}}
                                ])
[
  { _id: 'Bronx', count: 4 },
  { _id: 'Brooklyn', count: 14 },
  { _id: 'Manhattan', count: 35 },
  { _id: 'Queens', count: 58 }
]
```

Now you should be ready to answer some questions about the restaurants in the New York City restaurant data. Make sure to build your queries slowly, seeing the data before assuming it contains only the items you want. The most complex queries might take 4 or 5 test queries before you build the final query to answer the question and generate the results needed.

Step 2: New York City Restaurants Dataset

Write queries to answer the following questions:

1. (10 pts) How many distinct ***cuisine***s are represented in this collection?
2. (10 pts) Return the name of all restaurants within the ***zipcode*** 11105 which serves Greek ***cuisine***. Return only the ***name*** of each restaurant.
3. (10 pts) Find the borough with the most restaurants. Return the ***borough*** name and the number of restaurants.
4. (10 pts) How many restaurants in Manhattan have received a ***grade*** of "A" in all of their inspections?
5. (20 pts) List the top 5 cuisines with the highest average inspection scores. Return the ***cuisine*** and the average ***score***. (use ***avg*** and ***group*** functions)
6. (20 pts) Find the restaurant with the lowest total ***score***. Return the restaurant ***name*** and the total ***score***.
7. (20 pts) List the top 5 restaurants with the highest inspection scores. Return the restaurant ***name*** and the highest ***score***.