Lab 3: MongoDB queries and aggregations

Due date: Monday, January 28, 11:59pm.

Lab Assignment

Assignment Preparation

This is an individual lab. I expect every person to complete it without consulting others.

This is a two-part lab. The first part of the lab is relatively short and is simply designed to reinforce your familiarity with MongoDB's `db.collection.find()` command. The second part of the lab asks you to express a number of information needs as MongoDB aggregation pipelines.

Data

You will primarily be working with the Iowa Liquor sales data. I have created a MongoDB database `iowa`. The database has a number of collections in it, but the two collections you shall be using for this assignment are:

> `invoices`: the list of 10,000 invoices with all numeric attributes properly parsed.
>
> `populations`: the list of counties with population numbers properly parsed.

Essentially, these two collections contain the "correct" versions of the data. In the `invoices` collection you will find an `"_id"` attribute that uses the invoice Id as the unique Id of the record (the original `"Invoice/Item Number"` key is also present in the record). Additionally, all attributes that had values prefixed with the dollar sign in the original JSON (`"State Bottle Cost"`, "State Bottle Retail", "Sale (Dollars)") have these values converted to proper floating point values. Similarly, in the populations collection, the "_id" attribute replaces the original "id", the values of the "population" attribute are converted to proper integers (JSON had a comma in the values, and that meant parsing them as strings by default), and finally, the county name is stripped of the word "County" - this should matching county names between different collections more straightforward.

DO NOT USE ANY OTHER COLLECTIONS in your work.

db.collection.find() Queries

The main objective of this lab is for each of you to get comfortable using MongoDB's find() command. To that extent, you will write 20 MongoDB queries: 10 for each of the dataset.

Query preparation and submission. For each dataset, you will submit your queries in two separate ways:

1. **Text file.** Create text file lab3.mongo and and include all your queries there. Each query must be on its own lines, prefaced with a Javascript comment line specifying the query number and with at least one empty line between queries. The header of the file must contain one or more Javascript comment lines identifying with your name and other information about the file. The expected format is something like this:

```
// CSC 369. Lab 3.
// Alex Dekhtyar  -- you would be putting your name here
//

// Query 1
db.invoices.find(...)...

// Query 2
db.invoices.find(...)...

...

//end of queries
```

2. **Python program.** Create a python program queries.py that

   - accesses MongoDB through localhost connection,
   - reads in a file named credentials.auth (see below),
   - uses its contents to authenticate the MongoDB session,
   - switches to the iowa database, and
   - one by one runs each of your db.collection.find() queries, collects the output, and prints the output to terminal prefacing it with the query id and the query text.

The output of the program shall look roughly as follows:

```
Query 1:

db.invoices.find(...)...

Output:

{...
 ...
}
...
{...
...
}

Query 2
....
```

Query boundaries shall be easily viewable. Output itself may be either in the form of an array of JSON objects, or simply pretty printed JSON objects separated by new lines (I don't care about the proper JSON syntax in the output, but I don't mind it either - do whatever is easiest for you).

credentials.auth file. The `credentials.auth` file shalle contain a single JSON object describing the authentication credentials of a given user. The JSON object shall have the following format:

```
{ user: "<username>",
  pwd: "<password>",
  db: "<authentication database>"
}
```

For example, a file for used `bob` can look as follows:

```
{ user: "bob",
  pwd: "xyzw1234!",
  db: "admin"
}
```

This way, I can run your programs using my own credentials without having you to disclose yours to me.

Queries

Write MongoDB `db.<collection>.find()` queries that produce answers to each of the questions below. Each question must be answered with exactly one `find()` command.

1. Find all invoices for sales of alcohol from the `"VODKA 80 PROOF"` category, that resulted in over $1000 sale. For each invoice report its number (`"_id"` is enough) as well as the sale date, description of the specific item sold, number of bottles sold, and the total amount of the sale. Sort the output in descending order of the dollar amount of the sale. Pretty print the output.

2. Find the total number of the sales of `"PEACH BRANDIES"` and `"APPLE SCHNAPPS"` to stores in Polk County. Report just the number.

3. Find all instances (invoices) documenting the sales of more than 2 gallons of alcohol to `"H abd A Mini Mart Corp"` located in `"DES MOINES"` (city). Sort the invoices in descending order by the total amount of sale in dollars, and output the top three invoices. For each invoice keep the invoice number, the category of the spirit sold, name of the specific spirit (`"Item Description"`), number of bottles, and the total dollar amount of sale. Pretty print the output.

4. Find all sales by vendor named `"Diageo Americas"` in which over 100 bottles were sold. Report *only* the specific type of spirit (`"Item Description"`) for each sale[1]. Sort output in alphabetical order by the name of the spirit.

5. Find the most recent (latest by date) sale of more than 40 bottles to a store in one of the following counties: `"Linn", "Cerro Gordo", "Johnson", "Scott"`. Report the date, the name of the store, the county and city it is in, the description of the spirit purchased, number of bottles, and the total amount of sale in dollars. Pretty print the output.

6. Find the spirit sold to a store in Iowa with the largest retail price per bottle. Report the name of the spirit (`Item Description`) and the retail and state bottle costs [2]. Pretty print the output.

Aggregation Pipeline Queries

In this part of the lab, you will be replicating, as a MongoDB aggregation pipeline the seven tasks from Lab 1. For the sake of simplicity, the tasks are restated below.

Task 1. Compute the following information:

- Total number of bottles sold
- Total volume of alcohol sold in liters

[1]Do not worry about eliminating duplicates.

[2]With the data you have, there should be a unique answer spirit that matches the query.

- Overall revenue derived from all the sales
- Average price per bottle

This information has to be computed over **the entire dataset**, and reported back as a single JSON object with the structure (the field names are self-explanatory):

```
 {
   bottles: <number of bottles>,
   volume : <volume of alcohol in liters>,
   revenue: <overall revenue>,
   averagePrice: <average price per bottle>
}
```

Pretty print your output.

(Please note, this may take more than one aggregation pipeline step to accomplish.)

Task 2: We are interested in grouping some basic information about purchases made by each store into a single JSON document. Write an aggregation pipeline which produces **one JSON object per each individual store**. The object shall have the following format:

```
 { store: <number>,
   storeName: <store name>,
   purchases: [ {invoice: <invoice #>,
                  amount: <total dollar amount of purchase>}, ...
              ]
   dates: [ <date1>, <date2>,...]
}
```

Here, `store` and `storeName` are the `Store Number` and `Store Name` values from the original JSON documents; `purchases` contains an array of objects that document the invoice number and the `Sale (Dollars)` for each sale for that store, and `dates` is an array that stores **without duplication** all dates on which purchases were made by the given store. (Note: we, for this lab, are removing the requirement for the dates to be in chronological order. This is due to the fact that `Date` keys have string values in the format that is a bit difficult to properly sort outright.)

Task 3: We are interested in figuring out the distribution network for the wholesale alcohol sales. Write an aggregation pipeline that for each distributor (vendor) lists all stores to which it delivered. The format of the output is

```
{
 vendorId : <vendor number>,
 vendorName: <vendor name>,
 stores: [ {storeId: <store number>,
             store: <store name>,
             county: <county>
            }, ...
           ]
}
```

Please make sure that each store shows up exactly once in the array.

Task 4: We would like to find out all varieties of the category "VODKA 80 PROOF" that are being sold in the State of Iowa. Analyze the input dataset to discover such varieties and output them in the following format:

```
{type: "VODKA 80 PROOF",
 id: <unique id of the type of alcohol>,
 description: <name of the alcohol>
}
```

for each type of sold vodka with no duplicate entries. Note that the value of the `type` field is kept constant in your output, while `id` comes from the `Item Number` field in your original data, and `description` comes from the `Item Description` field.

(Note: you can substitute "_id" attribute name for "id" attribute name if this makes your query simpler to write.)

Task 5: We want to figure out how many individual sales (i.e., you are counting number of invoices) took place for each category of liquor sold in Iowa. Your output shall have the following format:

```
{
 catId: <category id>,
 category: <category name>,
 nSales: <number of sales>
}
```

The output shall contain one object per category, and shall be *sorted* in ascending order by category Id.

Preparing aggregation pipeline queries for submission

As with `db.collection.find()` queries, you will prepare the aggregation pipeline queries for submission in two ways.

Text file. Create a file `lab3-aggregation.mongo` with the same structure as your `lab3.mongo` file for `db.collection.find()` queries, and place the aggregation pipeline queries in it.

Python Program. Create a Python program `aggregation.py` that accesses MongoDB through `localhost` connection, reads the file named `credentials.auth`, uses its contents to authenticate MongoDO session, switches to the `iowa` database and runs one-by-one all aggregation pipeline queries, and prints their output, prefacing each query output with the text of the query and the query Id.

Submission

Submit the following artefacts:

- `README` file with your name and any comments regarding your submission.
- `lab3.mongo` and `lab3-aggregation.mongo` files containing your `db.collection().find()` and your aggregation pipeline queries respectively.
- `queries.py` and `aggregation.py` files.
- Outputs produced by `queries.py` and `aggregation.py` (redirect output to a file). Name the outputs `queries.out` and `aggregations.out`.

All code must have a title comment with your name in it.

Submit all your code in a single archive (zip or tar.gz).

Use `handin` to submit as follows:

```
$ handin dekhtyar lab3 <FILES>
```

Good Luck!