Lab 5: MongoDB Aggregation Pipelines

Due date: February 3, 11:59pm.

Please note: We are also moving Lab 4 due date to February 3, 11:59pm.

Lab Assignment

Assignment Preparation

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

This is a short lab to give you some familiarity with MongoDB's `db.<collection>.aggregate` command.

Data

In this lab you will build a number of MongoDB aggregation pipelines extracting and transforming data from some collections that you will set up. Below, we briefly describe the data collections you are expected to create.

You will use the collections you constructed for Lab 3 to test your aggredation pipelines.

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. The instructions are same as for Lab 3. For each dataset, you will submit your queries in two separate ways:

1. **Text file.** Create text files `beFuddled.mongo` and `thghtShre.mongo` 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
// BeFuddled dataset

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

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

...

//end of queries
```

2. **Javascript program.** Create a Javascript program that connects to the MongoDB server, runs, in turn, each of the queries, and prints out the results. The program shall connect to the database bearing your name, and use the prescribed collection name. For each query, the program shall print its number, the query itself, followed by the results obtained from running the query on the collection. Name your programs `beFuddled.js` and `thghtShre.js`.

BeFuddled queries

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

1. For each user report the number of games played, number of games won and number of games lost (note: games lost + games won may not add up to total number of games played, as there may be an incomplete game). Report output as the object[1]:

```
{ "user": <UserID>,
  "totalGames": <nTotalGames>,
  "won": <nGamesWon>,
  "lost": <nGamesLost>
}
```

[1] Unless specified explicitly, you may elect to keep the `"_id":` key in the output, or not keep it.

2. Report the top five moves that lead to the highest added points. The output should be the exact documents stored in the collection.

3. Report the final scores for the games in which the top five moves from the query above took place. The output format is

```
{
  "game": <gameId>,
  "user": <userId>,
  "score": <score>
}
```

   The final score is either the score of the game when it ended, or the score of the game after the last known move for the game was completed.

4. Find the 10 most popular regular moves. Report the moves sorted in descending order by frequency, in the following format:

```
{
  "location": { "x":  <x>,
                "y": <y> },
  "frequency": <numberOfOccurrences>
}
```

5. For each game create a single object representing all regular moves in the game. The object shall have the format:

```
{
  "game" : <gameId>,
  "user": <userId>,
  "moves": [<Move1>,..., <MoveN>]
  "score": <score>
}
```

   where each of `<MoveX>` values looks as follows:

```
 {
  "actionNumber": <actionNumber>,
  "location": { "x": <x>,
                "y": <y>
              }
 }
```

   and `<score>` is the *last* score of the game (the score at game end, or the score after the last known move of the game).

ThghtShre Queries

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

1. Report the message status with the largest number of messages in the collection. The output format is

   ```
   {
     "status": <status>,
     "messages": <Nmessages>
   }
   ```

2. For each user who sent out a message, produce a list of all *unique* recepients of their messages. A unique recepient is any of the common message recepients (`"all"`, `"self"`, `"subscribers"`, or any of the unique userIds that were the values of the `"recepient"` key for any message a user sent. The output format is

   ```
   {
    "user": <userId>
    "recepients": [<recepient1>, ..., <recepientN>]
   }
   ```

3. For each user who sent out a message, compute the total number of unique recepients. The output format is

   ```
   {
    "user": <userId>
    "recepients": <nRecepients>
   }
   ```

4. Report the message status (`"public"`, `"private"` or `"protected"` that had the largest number of messages addressed to `"self"`. The output format is

   ```
   {
    "status": <status>,
    "selfAddressed": <nSelfAddressedMessages>
   }
   ```

5. For each user who wrote more then two messages find the text of the second message they sent, and report it. The output format is:

   ```
   {
     "user": <userId>,
     "text": <messgeText>
   }
   ```

   Sort the output in ascending order by the user Id.

Submission

Submit the following artefacts:

- At least one javascript file creating a collection for each of the two datasets (see Lab 3 for instructions).
- `BeFuddled.mongo` and `ThghtShre.mongo` text files with queries.
- `BeFuddled.js` and `ThghtShre.js` Javascript programs with queries.
- `README` file.

All submitted files must contain your name on them.

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

Use `handin` to submit as follows:

Section 01:

```
$ handin dekhtyar lab5-01 <FILES>
```

Section 03:

```
$ handin dekhtyar lab5-03 <FILES>
```

Good Luck!