Lab 4: MongoDB Aggregation Pipelines

Due date: February 1, 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 in Lab 1 to test your aggredation pipelines.

Queries

The main objective of this lab is for each of you to get comfortable using MongoDB's `aggregate()` command.

Query preparation and submission. The instructions are same as for Lab 2. For each dataset, you will submit your queries in two separate ways:

1. **Text file.** Create text files `movieSurvey.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, properly authenticate you, 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 `movieSurvey.js` and `thghtShre.js`.

MovieSurvey 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()` pipeline. The pipeline itself can contain as many components as needed.

1. For each respondent, report their average movie rating. The return object shall have the following format:

```
{
 _id: <rid>,
 name: {first: <firstName>,
        last: <lastName>
       }
 avgscore: <avgScore>
}
```

2. Report all people who liked "Memento" more than "Dogma". Report the result in the format:

```
_id: <rid>,
name: {first: <firstName>,
        last: <lastName>
       },
memento: <MementoScore>,
dogma: <DogmaScore>
}
```

Sort the output in descending order of the rating of `"Memento"`.

3. Create an array consisting of all movie ratings for the `"Princess Bride"`. Report the result in a JSON object with the following format:

```
 {
  _id: "Princess Bride",
  ratings: [ <rating1>,...,<ratingN>]
 }
```

4. Report the name(s) of the person(s) who has/have the largest number of movie ratings above 8.0. The format of the object(s) to be returned:

```
{
 _id: <rid>,
 name: {first: <firstName>,
        last: <lastName>
       }
}
```

Note: This query requires a small, but a non-trivial hack. My version has eight steps in the pipeline. Two of these steps are `"$unwind"` steps. None of the steps are either `"$sort"` or `"$limit"`. Any query that involves a `"$limit": <N>` step is not going to work well, unless you can figure out an expression for `<N>` that turns it into the number of objects in the collection that share the largest number of ratings above 8.0. This is not impossible, but will essentially require similar trickery to what my query is doing.

5. Find the favorite movies (i.e., a movie with the highest average score) for each gender. For this exercise, it is ok to report just **one** of the movies with the highest score. For the query to be correct, the movie can be simply returned as the array index (0–12). For extra credit, turn it into an actual movie title (boring, but useful). Return results in the following format:

```
  { _id: <whatever>,
    gender: <gender>,
    favorite: <movieId>,
    score: <averageMovieScore>
  }
```

Here, we do not care what the value of the `_id` field is (it could be same as the value of the `gender` field) - this part is left up to you. `<gender>` is from `respondent.gender` field in the original object. `<movieId>` is either the movie index in the array or the movie title (for extra credit). `<averageMovieScore>` is the average score of this movie among people of same gender.

Note: My solution is a pipeline of nine operations. It involves grouping on a "compound" key, which is probably the single least intuitive part of the pipeline.

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).
- `movieSurvey.mongo` and `ThghtShre.mongo` text files with queries.
- `movieSurvey.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:

```
$ handin dekhtyar lab04 <FILES>
```

Good Luck!