# Homework 2 Answer Sheet

**Brian Detweiler**

## MongoDB: MovieLens User Ratings Queries

Note: For queries with many results, you may limit the results to a few records. This means, you don't have to show the complete result in your screenshot. You can use the documentation in MongoDB: <https://docs.mongodb.com/manual/>. With the exception of query 9, you should be able to answer all queries with the help of the tutorial in Homework 2 part 1.

Example:

Display the movie information for the movie with the ID 1.

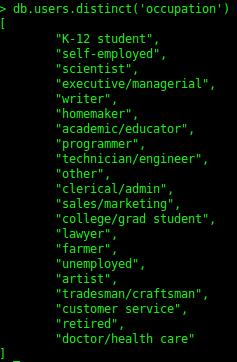
db.movies.find({\_id:1 }).pretty();



Now it's your turn:

1. Display all occupations. Each occupation should be displayed once.

db.users.distinct('occupation')



1. Chose an occupation and select all users with this occupation. Only show user information and hide the users' movie ratings.

db.users.find({occupation: "technician/engineer"}, {movies: 0})



1. Count the number of men in the database.

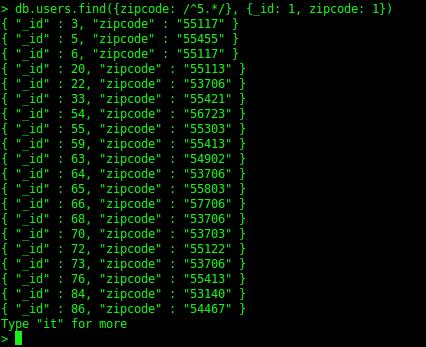
db.users.find({gender: 'M'}).count()

2717

homework2.2.3.jpg

1. Select all users whose zipcode starts with a 5. Only show the user ID and zipcode.

db.users.find({zipcode: /^5.\*/}, {\_id: 1, zipcode: 1})



1. Select all movies from the year 1998 and category comedy.

db.movies.find({year: 1998, category: 'Comedy'})



1. Count the number of movies from the year 1990 and 1995.

db.movies.find({year: {$gte: 1990, $lte: 1995}}).count()

1003

homework2.2.6.jpg

1. Display all movies published before the year 1992.

db.movies.find({year: {$lt: 1992}})

# homework2.2.7.jpg

1. Imagine that you registered for MovieLens. Create a new user with your user data. Do not include any ratings.

myself = {gender: 'M', age: '25-34', zipcode: '68102', occupation: 'technician/engineer'}

{

"gender" : "M",

"age" : "25-34",

"zipcode" : "68102",

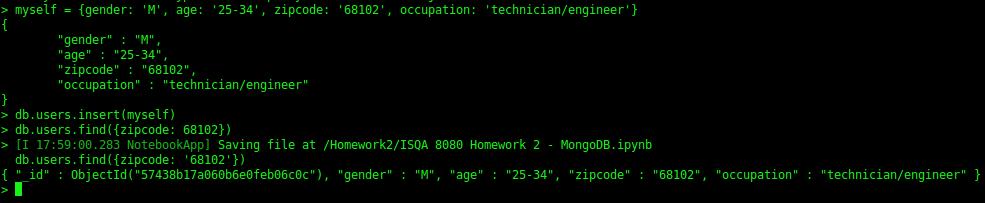
"occupation" : "technician/engineer"

}

db.users.insert(myself)

db.users.find({zipcode: '68102'})

{ "\_id" : ObjectId("57438b17a060b6e0feb06c0c"), "gender" : "M", "age" : "25-34", "zipcode" : "68102", "occupation" : "technician/engineer" }



**NOTE: I went with the auto-generated Mongo ObjectID, rather than finding the max \_id and incrementing it, since that is typically not good practice in Mongo.**

1. Update the user record you created in the previous query and insert a new rating for a movie of your choice. You will need to use $addToSet (<https://docs.mongodb.com/v3.0/reference/operator/update/addToSet/>) to add a value that does not exist in an array to the end of the array. You can use Math.round(new Date().getTime()/1000) to calculate the time in Unix format (seconds since start from epoch).

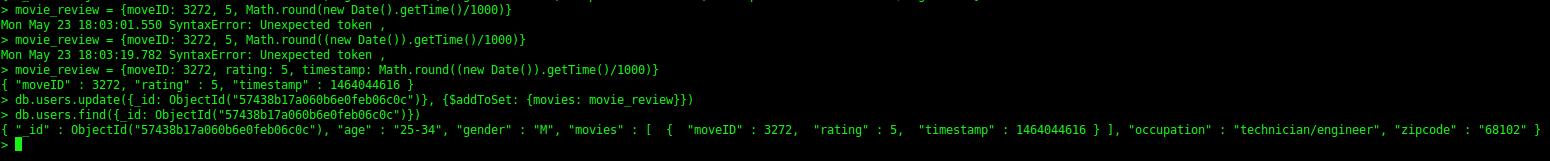
movie\_review = {moveID: 3272, rating: 5, timestamp: Math.round((new Date()).getTime()/1000)}

{ "moveID" : 3272, "rating" : 5, "timestamp" : 1464044616 }

db.users.update({\_id: ObjectId("57438b17a060b6e0feb06c0c")}, {$addToSet: {movies: movie\_review}})

db.users.find({\_id: ObjectId("57438b17a060b6e0feb06c0c")})

{ "\_id" : ObjectId("57438b17a060b6e0feb06c0c"), "age" : "25-34", "gender" : "M", "movies" : [ { "moveID" : 3272, "rating" : 5, "timestamp" : 1464044616 } ], "occupation" : "technician/engineer", "zipcode" : "68102" }



1. A query of your choice. **(Get all users near Omaha)**

db.users.find({zipcode: /^681.\*/}, {\_id: 1, zipcode: 1})

{ "\_id" : 385, "zipcode" : "68131" }

{ "\_id" : 389, "zipcode" : "68128" }

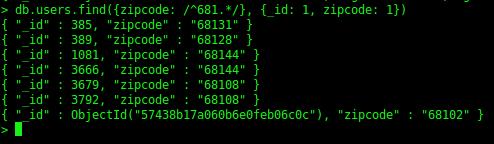
{ "\_id" : 1081, "zipcode" : "68144" }

{ "\_id" : 3666, "zipcode" : "68144" }

{ "\_id" : 3679, "zipcode" : "68108" }

{ "\_id" : 3792, "zipcode" : "68108" }

{ "\_id" : ObjectId("57438b17a060b6e0feb06c0c"), "zipcode" : "68102" }



# 

## Redis

Please describe your solution. Do not forget to include screenshots of the commands in Redis.

I am going to keep things simple with my solution and maintain two namespaces:

* **movieReviews**
  + **Keys are movie IDs**
  + **Values are hash maps of users and their reviews**
* **userMovies**
  + **Keys are User IDs**
  + **Values are sets of movies the user has reviewed**

In my example below, I'll use the Move IDs 1000, 1001, and 1002, and the User IDs 2000, 2001, and 2002. Each of the users has rated each of the movies in this example.

**movieReviews**

|  |  |
| --- | --- |
| **movieId** | **User Ratings** |
| 1000 | |  |  | | --- | --- | | **userId** | **rating** | | 2000 | 5 | | 2001 | 4 | | 2002 | 2 | |
| 1001 | |  |  | | --- | --- | | **userId** | **rating** | | 2000 | 3 | | 2001 | 3 | | 2002 | 3 | |
| 1002 | |  |  | | --- | --- | | **userId** | **rating** | | 2000 | 5 | | 2001 | 4 | | 2002 | 5 | |

**userMovies**

|  |  |
| --- | --- |
| **userId** | **Movies Rated** |
| 2000 | 1000, 1001, 1002 |
| 2001 | 1000, 1001, 1002 |
| 2002 | 1000, 1001, 1002 |

My application will ensure that when a new movie is rated, two inserts will be called:

**HSET movieRatings:{movieId} {userId} {rating}**

**SADD userMovies:{userId} {movieId}**

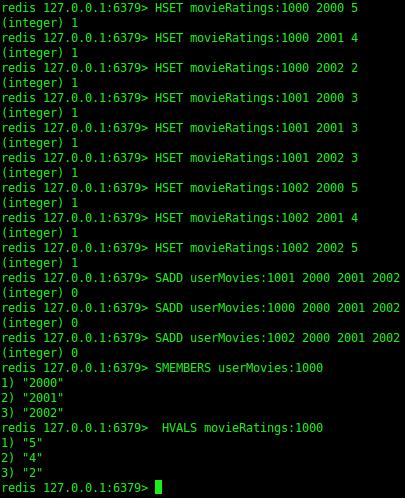
This ensures that whenever a user rates a movie, we are tracking both the rating of the movie, and the fact that the user has rated it.

We can retrieve the movies rated by each member using SMEMBERS:

**SMEMBERS userMovies:{userId}**

This covers requirements (1) and (3). For requirement (2) which asks me to average a movie's ratings, I would let the application handle this by pulling all reviews for a movie back and averaging them:

**HVALS movieRatings:{movieId}**



Redis Movie Review application proof of concept