Homework 2: MongoDB and Redis (Part 2)

Instructions: There are several tasks in this homework (not all will result in a deliverable). 3 tasks were explained in the first part. Please finish them before proceeding. The last tasks are explained in this part of the homework.

- The fourth task is to upload a dataset and to write and run different queries on this dataset.
- The fifth task starts with Redis. First, installation and running Redis either on your own computer or running it on the PKI Student Computers. You can find the directory Redis under App2.
- The sixth task demonstrates three examples of using Redis.
- The seventh tasks asks you to describe a solution to use Redis for storing user ratings.
- The eighth tasks asks you to write a reflection on the discussion boards.

Deliverables (Summary, Detailed description in the tasks):

- See task four: In the answer sheet "Homework 2 Answer Sheet.docx", write the 10 queries including a screenshot of the command and the first lines of the result.
- See task seven: In the answer sheet "Homework 2 Answer Sheet. Docx", describe how Redis can be used to store user ratings. You need to include screenshots to show how data would be stored as well as retrieved in Redis.
    - Submit the Answer Sheet on Blackboard using the link "Homework 2 Part 2: MongoDB and Redis" (where you downloaded the instructions and the text files).
- See task eight: Write a reflection by replying to the thread "Homework Reflection 2" in the discussion board "Homework Reflection".
- Due date Thursday, 5/26, 11:59pm. Due to the delay, no late submission penalty will be assessed.

Point Distribution and Grading Notes:

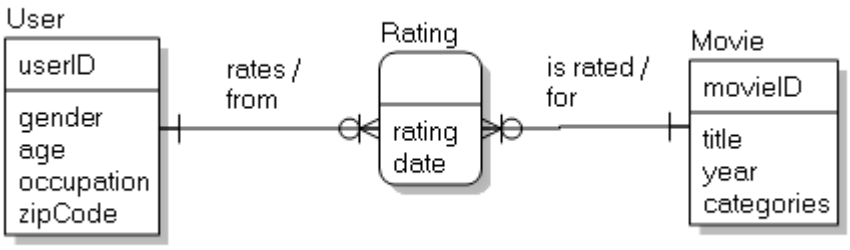
- Task four: Write down the 10 queries in the Answer Sheet including a screenshot for each of the 10 queries. Queries should be correct. (60 points)
- Task seven: A description in the Answer Sheet of how you would use Redis for processing and storing the user ratings for movies. Include a screenshots of commands that you will need to store and retrieve data. The solution does not need to be perfect but needs to address the functionality described in the task. (30 points)
- Task eight: Post your reflection in the required length. (10 points)

Task 4: MovieLens Dataset

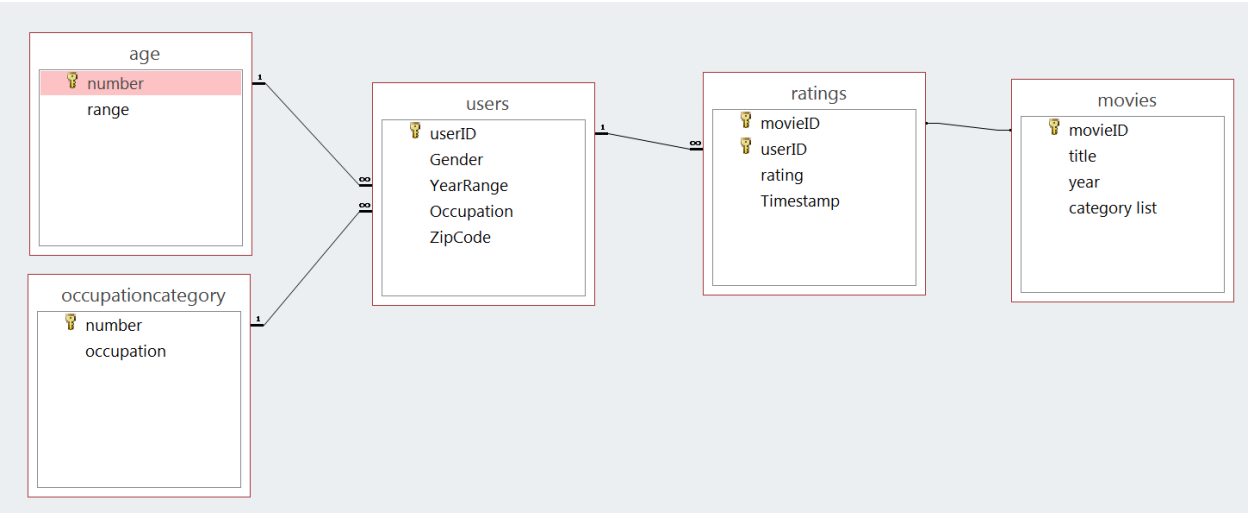
Please download the file users.txt and movies.txt (zipped, so do not forget to unzip them) from Blackboard (Homework 2 part 2) into you mongodb/bin directory. The data is from GroupLens Research (http://grouplens.org/datasets/movielens/) who collected data from the movielens.org website. I downloaded the 1M Dataset which contains 1 million ratings together with user and movie data. It is a small dataset that you will see in other homework assignments again. The data is stored in three text files using delimiters: user data (userID, gender, age range, occupation, zipcode), rating data (userID, movieID, rating, timestamp (in seconds from start of epoch), movie data (movieID, title with year, list of categories. I uploaded the data into an Access database, cleaned it (e.g., separating title and year, resolving zipcode format differences), and created two queries: (1) user and rating info, and (2) movie info. For 2000 ratings, no movie data was in the dataset. Since I am not aware of a nice tool that transforms tables with embedded documents into JSON format (tools do exists, but not sure whether they recognize embedded documents), I wrote a small parser that created two text files with the information in JSON format: users.txt and movies.txt. Conceptual and relational model as well as examples of the resulting JSON format below. Please notice that between user and rating, I chose to use embedded document to link user and rating together. For the link between rating and

movie, a reference (movieID) was used. Also notice, that the dataset used codes to represent age and occupation choices that do not exist in the JSON files. Also, technically we would need to have a m:n relationship between movie and category but instead category is a list (which is a violation of the properties of a relational table).

Conceptual Model (user or business view independent of a specific data model):



Relational Model (conceptual model translated to a relational model):



JSON user document example (movies is an array containing rating objects):

```
{
      "_id" : 1,
      "gender" : "F",
      "age" : "Under 18",
      "occupation" : "K-12 student",
      "zipcode" : "48067",
      "movies" : [
            { "movieID" : 6035,
               "rating" : 4,
               "timestamp" : 956712849},
            { "movieID" : 6032,
               "rating" : 4,
              "timestamp" : 956718127},
            { "movieID" : 6022,
              "rating" : 5,
              "timestamp" : 956755763}
            ]
      }
```

JSON movie document example (category is an array):

```
{
	"_id" : 1,
	"title" : "Toy Story ",
	"year" : 1995,
	"category" : ["Animation","Children's","Comedy"]
}
```

Please note that _id is the primary key for the document. If we were to use movieID, the MongoDB would create a primary key called _id as seen in the product example in part 1.

Now it is time for you to try out some queries on your own. In your mongo shell, create a new database (I called it movielensdb but you can call it whatever you want). After you downloaded both text files to your mongodb/bin directory (for me `c:/mongodb/bin`), use mongoimport to import each text file into the new database in mongodb. Use the command: `mongoimport -d <database name> -c <name of collection> --file <name of text file>` to upload each file into its corresponding collection. Use the collection names `users` and `movies` for the respective text file.

Queries

1. Display all occupations. Each occupation should be displayed once.
2. Chose an occupation and select all users with this occupation. Only show user information and hide the users' movie ratings.
3. Count the number of men in the database.
4. Select all users whose zipcode starts with a 5. Only show the user ID and zipcode.
5. Select all movies from the year 1998 and category comedy.
6. Count the number of movies from the year 1990 and 1995.
7. Display all movies published before the year 1992.
8. Imagine that you registered for MovieLens. Create a new user with your user data. Do not include any ratings.
9. 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).
10. A query of your choice.

Deliverables: Please use the word document on Blackboard (Homework 2 part 2 called "Homework 2 Answer Sheet.docx") to write down each query and paste a snapshot of the command and first lines of the result for each query. See the example in the word document "Homework 2 Answer Sheet.docx".

Aggregates

You may have noticed, that we did not any kind of queries that group records together. These are more complex (https://docs.mongodb.com/manual/aggregation/). E.g., counting the number of female and male in the set, we could write an aggregate pipeline: `db.users.aggregate ([ {$group : {_id: "$gender" , number : {$sum:1}}}]);`

Task 5: Key-Vaule Store Redis

Redis is an "open-source (BSD licensed), in-memory **data structure store**, used as database, cache and message broker. It supports data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs and geospatial indexes with radius queries." http://redis.io/

At address http://try.redis.io/ you can find an interactive tutorial, please go through the tutorial. It worked well with Chrome but not with IE for me. The tutorial goes through basic commands and will give you a feeling for some of basic data structures that are supported by Redis.

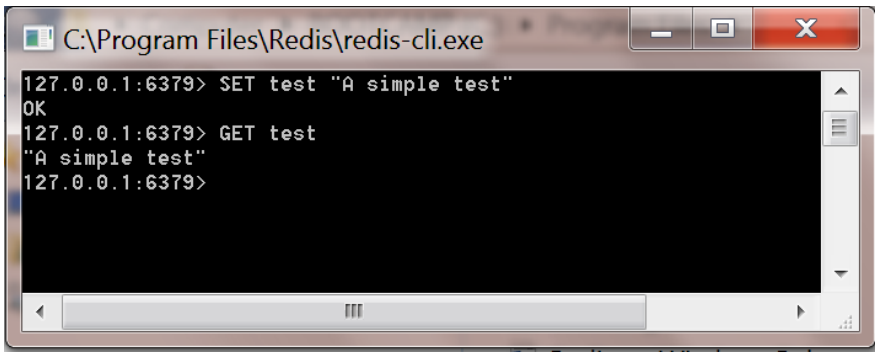
Deliverable: None for now. You can use the Redis Tutorial Commands.docx (on Blackboard under Homework 2 Part 2) to see what commands you covered.

Task 6: Installing and Running Redis

So, now you got some ideas what you can do with the database. Only entering commands into Redis though is not enough. Usually you would use an application that connects to the databases and sends commands to the database. In order to not complicate the homework, we will not do so but directly enter the commands into Redis.

This homework will explain the installation for a Windows installation. You should be able to find enough sources to easily setup Redis for Linux or OS X. If you do not want to use your own laptop, please use the PKI computers. You should find Redis in the App2 directory.

1. Go to https://github.com/MSOpenTech/redis/releases and download the .msi for Redis. It only supported a 64bit system. Download the most recent stable release with bug fixes (for me it is 3.0.501).
2. Run the .msi file and follow the installation steps. I used the default settings.
3. Go to the folder Redis is installed and look over the word fildes (release notes, info on Redis on Windows, windows service documentation). For me the folder is `c:/Program Files/Redis/`.
4. If you installed Redis using the .msi Redis is already running as a Service. You may have to reboot your computer, it was necessary for me.
5. Double-click on the file `redis-cli.exe`.
6. A command line console should open. Test Redis.



7. You can now use Redis similar to the tutorial and enter commands directly to enter or manipulate data in Redis. `HELP command` gives you help on commands.
8. Did it work? If yes, then you are good to go. If not, try to find out your errors or post on the discussion boards in Blackboard in order to receive help.

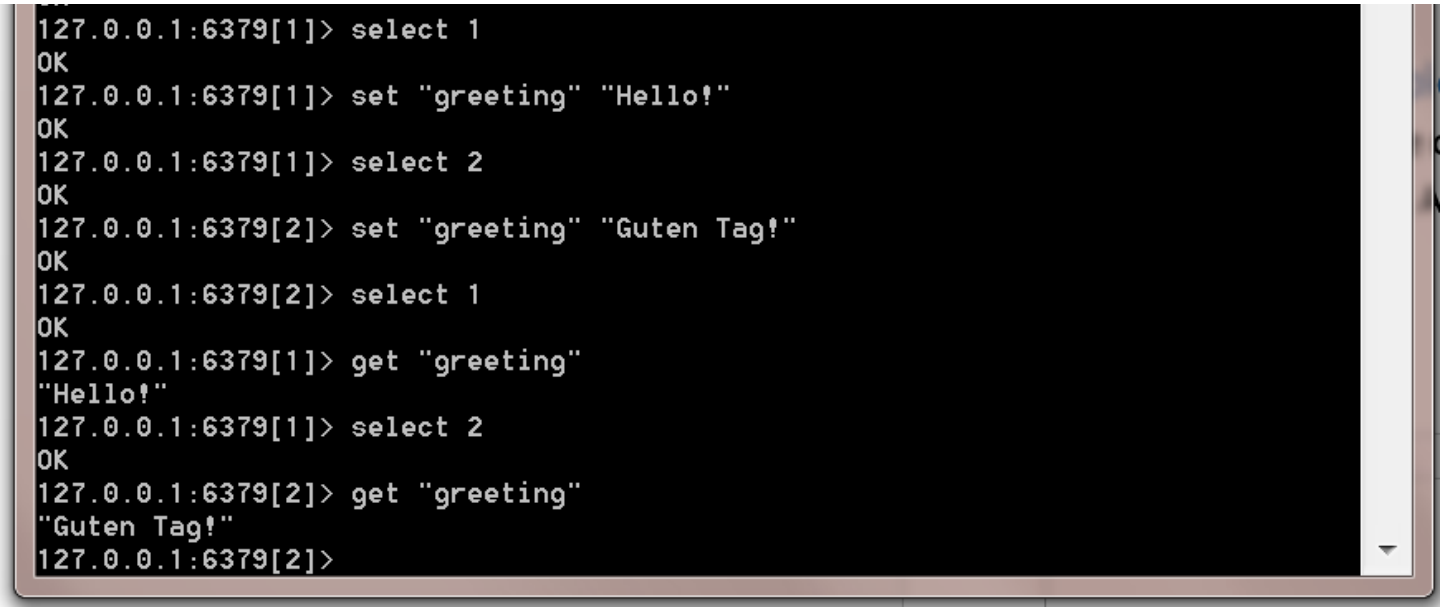
Deliverable: None.

Task 7: Going through some Examples

Now that we are set up, we can go through some examples to get an idea how Redis can be used. These examples are adapted from “NoSQL for mere Mortals” and “Redis Essentials”. Please note that some examples do require programming experience, so that there is a limit to what we can do. If you want to learn more, Redis Essentials or another Redis book are good starting points. You can get them from the UNO library. There are several available online. You can also try to use your favorite programming languages and IDE and write more complex application that use Redis as a database.

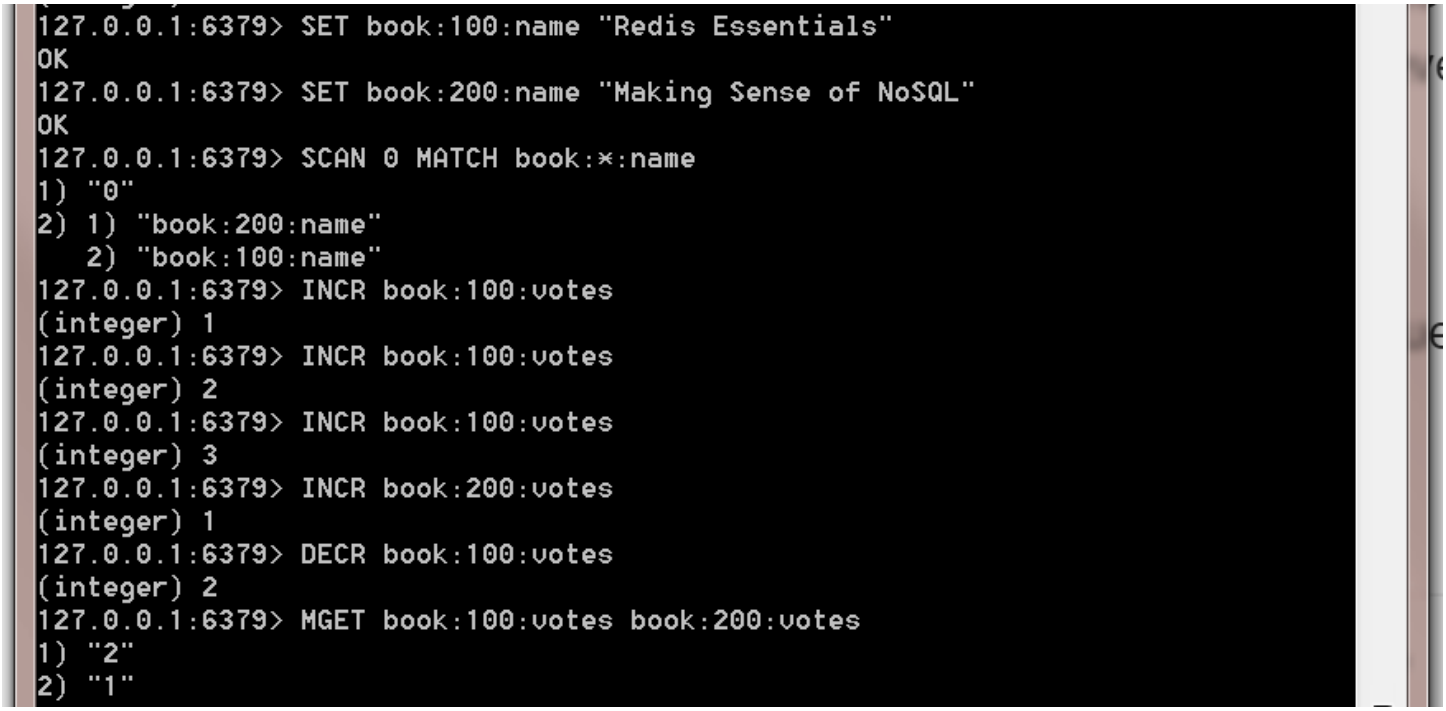
Example 1: Example Namespaces

1. This example creates two namespaces (also called databases), one for English and the other one for German expressions. An application could get the correct language term by selecting the databases but sending the same key. As you can see in the following dialogue, we first select (or create if it does not exist) a new namespace and enter a new key-value pair. We then switch to the second namespace and enter the value with the German translation and the same key as used before. If you do not change to the second namespace, but use the first one, you will override the key-value entry. An application could now easily switch between the two languages, by setting the namespace to either English (1) or German (2) and send the same key to get the correct word.



Example 2: Building an book voting system

1. This example allows to up and down vote books. The first two SET commands will enter the names of two books. SCAN allows to search for keys. The example shows how it searches for keys that start with book and end with name. The show the keys for the books we just entered. The next five commands increment or decrement the votes for each book. Do note the naming conventions <entity><entity ID><attribute> that we are using. This pattern allows us to set the name and then the votes for the two books. The last command returns the resulting votes for the books. Again, you need to imagine this example in connection with an application or a website.



Example 3: Leaderboard

1. This example creates a leadership board and adds user to it. It returns the top players. We use the advantages of a sorted list here. The first commands add to the leadership board called game-score the players with their score. We then display the top five players with their score. The last command displays the rank of player Sue.

```
127.0.0.1:6379> ZADD "game-score" 30 "Tim"
(integer) 1
```

```
127.0.0.1:6379> ZADD "game-score" 23 "Sue"
(integer) 1
127.0.0.1:6379> ZADD "game-score" 3 "Mary"
(integer) 1
127.0.0.1:6379> ZADD "game-score" 45 "Joe"
(integer) 1
127.0.0.1:6379> ZADD "game-score" 21 "Bob"
(integer) 1
127.0.0.1:6379> ZADD "game-score" 14 "Ron"
(integer) 1
127.0.0.1:6379> ZADD "game-score" 16 "Tom"
(integer) 1
127.0.0.1:6379> ZADD "game-score" 25 "Lisa"
(integer) 1
127.0.0.1:6379> ZREVRANGE "game-score" 0 4 WITHSCORES
 1) "Joe"
 2) "45"
 3) "Tim"
 4) "30"
 5) "Lisa"
 6) "25"
 7) "Sue"
 8) "23"
 9) "Bob"
10) "21"
127.0.0.1:6379> ZREVRANK "game-score" "Sue"
(integer) 3
127.0.0.1:6379>
```

2. Of course, during a game, the score would change. We can easily change the score of a player by overwriting it.

```
127.0.0.1:6379> ZADD "game-score" 48 "Tim"
(integer) 0
127.0.0.1:6379> ZREVRANGE "game-score" 0 4 WITHSCORES
 1) "Tim"
 2) "48"
 3) "Joe"
 4) "45"
 5) "Lisa"
 6) "25"
 7) "Sue"
 8) "23"
 9) "Bob"
10) "21"
127.0.0.1:6379>
```

Deliverable: We want to use Redis to store the user ratings for our movies. Please describe a possible solution of a website or application that can (1) store a rating every time a user rates a movie, and (2) display the average rating for each movie, and (3) display for one user all ids a user rated. Please remember that Redis is used together with an application, so certain tasks could be done by the application instead of Redis (e.g., certain claculations). Please describe what Redis does and what the application.

There will be different ways to implement this in Redis. Please describe your solution. Your solution does not need to be perfect, but should somehow achieve the tasks above. In addition to the description, please take a screenshot of the commands you would need to use to get data into or out of Redis (similar to what I have done above).

Task 8: Reflection

Write a reflection (200 to 250 words) to the discussion board. Reflect on your experience going through the homework. You can write about whatever you like. Likes, dislikes, challenges, criticism, your thoughts on MongoDB, potential uses of MongoDB, open questions, etc. You can also address points or questions from other students' reflections, if you like.

Deliverable: Write and post your reflection to the discussion board "Homework Reflections" in the "Homework 2 Reflection" thread. Reply to this thread. Do not start your own thread.