TP MongoDB

1 Basic Concepts

MongoDB is a NoSQL, document-oriented database system. As such, it moves away from traditional relational databases. A database consists of a set of collections which, in turn, store documents with a dynamic, not-predefined schema.

A document is an ordered set of key-value pairs, where keys are strings and values can be any of the defined data types (null, boolean, number, string, date, regular expression, array, object id, binary data and code), or another document itself, called *embedded document*. Every document has a special key `_id` which is unique in the collection. However, documents within a collection may follow different schemas, i.e., consist of different sets of key-value pairs.

2 Install & Run MongoDB

1. Download your OS version of MongoDB on this page : `http://www.mongodb.org/downloads`.
2. Extract the content of the archive in a folder of you choice (`$MONGO_HOME` in the following).
3. Create a directory for storing the database files, e.g., `/path/to/db/dir`.
4. Run the server and wait until the initialization is finished.

```
# $MONGO_HOME/bin/mongod --dbpath /path/to/db/dir
...
<Date> [initandlisten] waiting for connections on port 27017
```

   You need to keep this program running and use a different terminal for the next commands. Alternatively, you can use the options `--fork` and `--logpath /path/to/logfile`. In this case, the server is started as a background process and the output is sent to the specified log file.

   In order to shutdown the server you have to execute the following commands in the MongoDB shell:

```
> use admin
switched to db admin
> db.shutdownServer()
server should be down...
```

5. Download the following files:
   - `http://www.lirmm.fr/~servajean/movielens_movies.json`
   - `http://www.lirmm.fr/~servajean/movielens_users.json`
6. Import the two downloaded files into MongoDB:

```
# $MONGO_HOME/mongoimport --db MovieLens --collection movies
  --file movielens_movies.json
connected to: 127.0.0.1
```

```
<Date> check 9 3883
<Date> imported 3883 objects
# $MONGO_HOME/mongoimport --db MovieLens --collection users
  --file movielens_users.json
connected to: 127.0.0.1
<Date> check 9 6040
<Date> imported 6040 objects
#
```

With these commands we are creating the collections `movies` and `users` in the database `MovieLens` and populating them with the data stored in the `json` files.

7. Connect to the MovieLens database:

```
# $MONGO_HOME/bin/mongo MovieLens
MongoDB shell version: 2.4.8
connecting to: MovieLens
>
```

3 MongoDB Shell

The last executed command starts the MongoDB shell, which is in fact a full-featured JavaScript interpreter, where you can run arbitrary JavaScript code. Moreover, it provides a set of add-ons analog to those present in other database shells, for instance:

- `use <db_name>`: Connects to the database `<db_name>`.
- `show collections`: Display the collections of the current database.
- `help`: Show information about the most important commands.

4 Database Schema

As explained before, documents within a collection do not necessarily follow the same schema. However, in the example used for the questions, documents of each collection have a similar data structure. We give an example of each:

The collection `movies` contains information about the movies, including their id, title (which includes the year of production) and genres. This is not the best representation, as you may discover while answering the questions. In Section 7 we will take care of that.

```
> db.movies.findOne();
{
        "_id" : 1,
        "title" : "Toy Story (1995)",
        "genres" : "Animation|Children's|Comedy"
}
```

The collection `users` contains information about the users and the ratings they have given. User information includes his/her id, name, age, occupation and sex. The ratings are included as an array of embedded documents, each one having a movie id (which references one of the

movies in the other collection), a rating and a timestamp which indicates when the rating was included). Note that the timestamp does not follow JavaScript standard, as it indicates the number of seconds since the Unix epoch, instead of the number of milliseconds as in Javascript. We will modify this representation in Section 7.

```
> db.users.findOne({},{movies : {$slice : 2}});
{
        "_id" : 6040,
        "name" : "Barry Erin",
        "gender" : "M",
        "age" : 32,
        "occupation" : "doctor/health care",
        "movies" : [
                {
                        "movieid" : 573,
                        "rating" : 4,
                        "timestamp" : 956704056
                },
                {
                        "movieid" : 589,
                        "rating" : 4,
                        "timestamp" : 956704996
                }
        ]
}
```

5 Basic Operations

5.1 Read queries

Question 1
How many users are there in the database?
Ref: `http://docs.mongodb.org/manual/reference/command/count/`.

Question 2
How many movies are there in the database?

Question 3
Which is the occupation of *Clifford Johnathan*? In the answer show only its name and occupation.
Ref: `http://docs.mongodb.org/manual/reference/method/db.collection.find/`.

Question 4
How many users are there between 18 and 30 years old (both included)?

Question 5
How many users are *artist* or *scientist*?

Question 6
Who are the 10 oldest woman writers?

Question 7
Show all the occupations in the database.

5.2 Inserts, Updates and Deletes

Question 8
Insert yourself in the database but do not include the field `movies`. Do not worry about privacy, you are not obliged to use real information.

Question 9
Select one movie from the collection `movies` and update your entry by adding the field `movies`, which is an array, including a review of the film.
Hint: You can use the current timestamp : `Math.round(new Date().getTime() / 1000)`.

Question 10
Remove your entry from the collection.

Do not worry if you accidentally removed other entries from the collection `users`. You can always recreate the original one by removing all its documents and reimporting them with `mongoimport`.

Question 11
Change in all users the occupation *programmer* by *developer*.

6 Regular Expressions

Question 12
How many movies have been released in the 80s?
Hint: The titles include the released date in parentheses.
Ref: `http://docs.mongodb.org/manual/reference/operator/query/regex/`.

Question 13
How many movies have been released between 1984 and 1992?

Question 14
How many horror movies are there?

Question 15
How many movies are both *Musical* and *Romance*?

7 ForEach

Question 16
As you have just seen, having the year integrated in the title name is not very practical. Modify the collection `movies` by creating a new field called `year` and removing it from the title.
Ref: `http://docs.mongodb.org/manual/reference/method/cursor.forEach/`.
Hint: You may use the command `replace` to select the parts of the title that you need.

Remark: MongoDB stores documents of a collection as list, one after the other. Between documents a padding is included so that changes in the size of a specific document can be done in place. However, if the document size increases more than the assigned padding, it it is relocated at the end of the list. When a normal `find()` is executed in MongoDB, documents are not retrieved all at once but in batches. As a consequence, if when accessing a collection, we modify a document and it has to be relocated at the end, it may be returned twice. In order to avoid this situation, MongoDB provides the command `snapshot()` which traverses the `_id` index. However it makes queries slower, so it should only be used when needed.
Ref: `http://docs.mongodb.org/manual/reference/method/cursor.snapshot/`

Question 17
Modify the collection `movies` by replacing the string field `genres` with an array that contains all the genres.

Question 18
Modify the collection `user` by replacing the field `timestamp` with a new field called `date` of type `Date`.
Hint: Field `timestamp` in the original dataset is in seconds since the Unix epoch, but dates in Javascript can be created by using the number of milliseconds since the Unix epoch.
Hint: You can use `delete Object.attribute` to delete an attribute from an object. Another possibility is using the option `$unset` from the `update` command.

8 Queries on Arrays

8.1 Read Queries

Question 19
How many users have seen the movie with `_id` 1196 (*Star Wars: Episode V - The Empire Strikes Back (1980)*)?

Question 20
How many users have seen all the movies from the old Star Wars trilogy (IV,V,VI) with ids: 260, 1196, 1210?
Ref: `http://docs.mongodb.org/manual/reference/operator/query/all/`.

Question 21
How many users have rated exactly 48 movies?
Ref: `http://docs.mongodb.org/manual/reference/operator/query/size/`.

However, `$size` can only match exact number. Selecting users that have rated more than a specified number of movies has to be done in two steps which will be the subject of the following questions.

Question 22
For each user create a field `num_ratings` that display the number of ratings he has given.

Question 23
How many users have rated more than 90 movies?

Question 24
How many ratings have been submitted after 1st January 2001?

Question 25
Which are the last 5 films rated by *Jayson Brad*. Do not use the date but the order in which they have been added to the array `movies` (assume that ratings are added to the end).

Question 26
Return only the information about *Tracy Edward*'s information about the rating of the film *Star Wars: Episode VI - Return of the Jedi* (whose id is 1210).

Question 27
Find how many users have rated the movie *Untouchables, The* with a note of 5.

8.2 Update Queries

Question 28
Barry Erin has just seen the movie *Nixon* with id 14.. Add the movie with a rating of 4 in the array `movies` of *Barry Erin*. Recall that the field `num_ratings` shuld reflect the size of the movies' array.
Hint: `http://docs.mongodb.org/manual/reference/method/db.collection.update/`.

Question 29
Marquis Billie does not assume having seen the movie *Santa with Muscles* (whose id is 1311). For her dignity, please remove this movie from her movies' list.

Question 30
The genres for the film *Cinderella* should be *Animation*, *Children's* and *Musical*. Modify the document so that it has the three genres with no duplicates in just one query.

9 References

Joins are not supported in MongoDB. Normally we denormalize the references and store them as embedded documents, but some times it is better to store related information in different documents of different collections and then add a reference. There are two ways of doing that:

- *Manual references:* We store the field `_id` of one document in another document. This is the case of our example, where the `_id` of the movies is stored in in the ratings array.
- *DBRef:* A DBRef is a convention to represent a document. It includes the information of its collections, its id and optionally, the database where it is stored. Some drivers leverage on this information and are capable of resolving automatically a DBRef. However, it is discouraged to use them as some operations do not support DBRefs.

Question 31
Modify the collection `users` and add a new field `movies.movieref` that stores a DBRef to the movie.
Ref: `http://docs.mongodb.org/manual/reference/database-references/`.

Question 32
Find how many users have rated the movie *Taxi Driver* using the new created field.

Question 33

Find how many users have rated the movie *Taxi Driver* with a note of 5 using the new created field.

After executing the query, you may want to remove the field `movieref` as it is not going to be used anymore. Use the following code:

```
> db.users.find().forEach(function(u) {
    for (i = 0; i < u.movies.length; i++) {
        delete u.movies[i].movieref;
    }
    db.users.save(u);
});
```

Note that it is not possible to update all the elements in an array with an `update()`.

10 Indexes

Question 34

Given the name of the 10 women that have rated a film most recently. Notice that if you add the `explain()` function at the end of your query, you will have some statistics about its execution.

Question 35

Create an index on both `gender` and `movies.date` simultaneously.
Ref: `http://docs.mongodb.org/manual/reference/method/db.collection.ensureIndex/`.

Question 36

Re-execute question 34 with the explain function. What happened?

11 Aggregate Queries

Question 37

Show how many films have been produced on each year of the 90s and sort them is descending order of number of films.
Ref: `http://docs.mongodb.org/manual/core/aggregation-pipeline/`.
Ref: `http://docs.mongodb.org/manual/reference/operator/aggregation-pipeline/`.
Remark: This query is much easier if you have done Question 16.

Question 38

Which is the average rating of the film *Pulp Fiction* (whose id is 296)?

Question 39

Return for each user and in one query his/her id, name, maximum, minimum and average rating he has given to his movies and sort them in increasing number average note.

Question 40

Which has been the month where more ratings have been added to the database?

Question 41

Create a new collection `join` that associates for each movie its `_id`, its `title`, its `genres`, its `year` and all ratings the users have given to the movie.
Hint: `aggregate` + `insert` + `forEach`.

12 Map-Reduce

As an alternative to the *Aggregation Pipeline* you can use Map-Reduce to perform the aggregations. Normally the former delivers better performance, but Map-Reduce is much more flexible and allow to write arbitrary code. You can find more information in the following reference:
Ref: `http://docs.mongodb.org/manual/core/map-reduce/`.

Question 42

What is the most popular genre in number of ratings?
Hint: Use the `join` collection created in Section 36, question 41.

Question 43

What is the most well rated genre (best rating average)?
Hint: Use `finalize`.

Question 44

Show, for each year, the movie that has received the highest number of ratings.

Question 45

Show, for each year, the movie that has received the highest ratings values in average.

Question 46

Show, for each year, the movie that has received the highest ratings values in average only takings movies that have been rated at least 1000 times.
Hint: Use the `query` option of Map-Reduce to filter documents before executing the functions. You may need to do something on the collection `join` before.