Data Management and Visualization

Politecnico di Torino

Create and query a MongoDB collection – Practice 5

Goal

The objective of the practice is to connect to a MongoDB instance, create and successfully populate a collection of documents. Then, visually explore the newly created collection and query the database exploiting different MongoDB functionalities and patterns.

Initial set-up

At LABInf:

Windows:

1) Create a local folder (e.g.: `C:\Users\<S123456>\Desktop\`*`mongo_database`*) and save its path, from now on called: *my_database_path*.
2) Navigate to `C:\Program Files\MongoDB\4.0\bin` and open a command shell in the location (maiusc + right-click -> open command window here).
3) Run the following command:
   ```
   > mongod --dbpath my_database_path
   ```
4) Open another Command Shell in the previous location (`C:\Program Files\MongoDB\4.0\bin`).
5) Run the following command:
   ```
   > mongo
   ```
   You are now logged into the Mongo Shell.

At home:

Before creating the database collection, we need a running MongoDB instance. To do that, we will user the Docker Platform to create a service running MongoDB and Mongo-Express.

Linux requirements: Docker, Docker Compose
Windows requirements: Docker Toolbox for Windows 10 or older.
(for Windows 10 Pro or Enterprise download and run Docker Desktop)

Place the Docker Compose file (***docker-compose.yml***, provided with practice) in a folder (e.g., $HOME/MongoDB), open a terminal and run the following commands (if the Docker Containers do not start, try renaming the file ***docker-compose.yml*** into ***docker-compose.yaml***):

```
docker-compose up -d
docker-compose ps
```

Copy the name of the docker running MongoDB (from now on called: name_docker_mongo).

```
docker exec -it name_docker_mongo bash
```

Login into the Mongo Shell:

```
mongo admin -u root -p example
```

You are now logged into the Mongo Shell.

In order to connect to MongoDB Express:
Linux: connect to *localhost:8081*
Windows: in the Docker Interactive Shell, run the command: docker-machine ip default, then connect to the retrieved IP address at port 8081. Alternatively, access *Kitematic* (installed along with Docker Toolbox), go to the mongodb_mongo_express container and look-up the Docker IP address and port.

Creating the database collection

1) Create a new database for the practice. (`use restaurants`)
2) Populate the newly created database by inserting the documents in the **restaurants_collection.txt** (download it from the course website).
   use the command: `db.restaurants.insertMany(<file content>)`
3) In order to check the success of the insert, run the command:
   `db.restaurants.find().pretty()`
4) Connect to MongoDB Express to visually explore the dataset (only for the Home set-up).

Running queries of interest

Run the following queries of interest:

1) **Find all restaurants whose cost is *medium***
2) **Find all restaurants whose review is bigger than 4 and cost is *medium* or *low***
3) **Find all restaurants that can contain more than 5 people and:**
   a. **whose tag contains "italian" or "japanese" and cost is *medium* or *high* OR**
   b. **whose tag does not contain neither "italian" nor "japanese", and whose review is higher than 4.5**
4) **Calculate the average review of all restaurants**
5) **Count the number of restaurants whose review is higher than 4.5 and can contain more than 5 people**
6) **Run query n. 4 using the Map-Reduce paradigm**
7) **Run query n.5 using the Map-Reduce paradigm**
8) **Find the restaurant in the collection which is nearest to the point [45.0644, 7.6598]**
   Hint: remember to create the geospatial index.
9) **Find how many restaurants in the collection are within 500 meters from the point [45.0623, 7.6627]**

Queries' solution and expected output

1) **db.restaurants.find({"cost":"medium"}).pretty()**
   -> 001, 009, 010

2) **db.restaurants.find({review:{$gt:4},$or:[{cost:"medium"},{cost:"low"}]}).pretty()**
   -> 001, 002, 006, 008, 010

3) **db.restaurants.find({$or:[{$or:[{tag:"italian"},{tag:"japanese"}],$or:[{cost:"medium",cost:"high"}]},{tag:{$ne:"italian"},tag:{$ne:"japanese"},review:{$gt:4.5}}],maxPeople:{$gt:5}})**
   -> 004, 006, 007

4) **db.restaurants.aggregate([{$group:{_id:null,review_avg:{$avg:"$review"}}}])**
   -> { "_id" : null, "review_avg" : 4.26 }

5) **db.restaurants.aggregate([{$match:{review:{$gt:4.5},maxPeople:{$gt:5}}},{$group:{_id:null,count:{$sum:1}}}])**
   -> { "_id" : null, "count" : 2 }

6) **db.restaurants.mapReduce(function() {emit(null, this.review);},function(key, values) {return Array.avg(values);},{out:"restaurants_reviews_avg"})**

7) **db.restaurants.mapReduce(function() {emit(null, 1);},function(key, values) {return Array.sum(values);},{query: { review:{$gt:4.5},maxPeople:{$gt:5}}, out: "restaurants_reviewsGT5_maxPGT5"})**

8) **db.restaurants.createIndex({location: "2dsphere"})**
   db.restaurants.findOne({location:{$near:{$geometry:{type: "Point",coordinates: [45.0644,7.6598]}}}})
   -> 006
9) **db.restaurants.createIndex({location: "2dsphere"})**
   db.restaurants.find({location:{$near:{$geometry:{type: "Point",coordinates: [45.0623,7.6627]},$maxDistance: 500}}}).count()

   -> 3