Data Management and Visualization

Politecnico di Torino

Create and query a MongoDB collection – Practice 5

Goal

The objective of the practice is to create and successfully populate a collection of documents. Then, query the database exploiting different MongoDB functionalities and patterns. Optionally, configure a replica set in a Docker environment and connect to the MongoDB database.

Database Connection

Remote database connection

1. Download MongoDB Compass at https://www.mongodb.com/try/download/compass
2. Install and open the application.
3. Create a free cluster (if you do not have already one)
    a. create a MongoDB account (https://www.mongodb.com/cloud/atlas/register)
    b. Select *"Shared Cluster"* option (free cluster). The default settings are the ones set to get the account completely free.
    c. give a name to your cluster
    d. go to Database Access
        i. click on *"Add New Database User"*
        ii. select authentication method with password
        iii. fill in the form
        iv. leave all the default options
    e. configure remote access
        i. click on *"Network Access"*
        ii. click on *"Add IP Address"*
        iii. enter in the *Access List Entry* field 0.0.0.0/0
4. Get string connection from MongoDB Atlas server
    a. go on the "*Database Deployments"* page from the side menu
    b. click *"Connect"* near the newly created cluster
    c. select "*Compass*"
    d. copy the connection string
5. Paste the connection string into MongoDB Compass
6. Click on **Connect**

Create a database and populate using Python

1) Install pymongo on the local machine using the following shell command

```
python -m pip install pymongo
```

2) Download the script from the webiste (`create_database.py`)
3) Get the connection string from MongoDB Atlas
4) Modify the connection string inside the script file
5) Run the script using the following command

```
python create_database.py
```

6) Check the database using MongoDB Compass

Running queries of interest

Each document of the collection has a structure with the following fields:

```
{_id: <ObjectId>,
name: <string>, // name of the restaurant
tag: <list[string]>, // tags assigned by the users
orderNeeded: <boolean>, // if the user should reserve
maxPeople:<int>, // maximum number of customers
review:<float>, // average vote
cost:<string>, // classification of the menu price. Categories are: low, medium and
high
location:{type:"Point",coordinates:[<lat>,<long>]}, // geographical point
contact:{
      phone:<string>, // telephone of the restaurant
      facebook:<string> // link to the facebook page
      }
}
```

Run the following queries of interest:

1) Find all restaurants whose cost is medium. Show the result is the pretty format.
2) Select the name and the number of seats (maxPeople) available of all the restaurants whose review is bigger than 4 and cost is medium or low
3) Select the name, the phone of the 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

    Remove from the output the field _id.
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) Find the restaurant in the collection which is nearest to the point [45.0644, 7.6598]

    Hint: remember to create the geospatial index.
7) Find how many restaurants in the collection are within 500 meters from the point [45.0623, 7.6627]
8) Add the tag “pizza” to all the restaurants that contain the tag “italian”. If the tag “pizza” is already present, you should not insert it.

9) Decrease the review score of 0.2 for all the restaurants with the tag 'fastfood'
10) For only the restaurants with a review higher than 3, find the tags which appear more than 1 time. For each tag, show how many documents include it.
11) For each cost category, compute the minimum review rate, the maximum review rate, the average review rate and the number of restaurants. Sort the result in descending order according to the number of restaurants in each cost category.
12) Find the median value of maxPeople attribute.

Replica set (Bonus)

Initial set-up

To create a replica set we will use the Docker Platform.

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 command (if the Docker Containers do not start, try renaming the file **docker-compose.yml** into **docker-compose.yaml**):

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

The docker services will run in detached mode. If you want to see the output of each container, remove the option -d form the previous command.

Configure the replica set

The docker-compose.yml file defines 3 instances of MongoDB. The name of each instance can be retrieved using the command:

```
docker ps
```

All the following commands should be launched inside the folder of the project (e.g., $HOME/MongoDB).

1) Connect to the Mongo shell of one instance using the following command

```
docker-compose exec name_docker_mongo mongo
```

   where *name_docker_mongo* is the name of one mongo instance.

2) Configure the replica set with the following requirements:
   a) replica set name should be equal to *rspoli*
   b) the priority of polimongodb1 instance should the highest one
   c) the priority of polimongodb3 instance should the lower one

```
rsconf = {
   _id : "rspoli",
   members: [
       {
           "_id": 0,
           "host": "polimongodb1:27017",
           "priority": 4
       },
       {
           "_id": 1,
           "host": "polimongodb2:27017",
           "priority": 2
       },
       {
```

```
            "_id": 2,
            "host": "polimongodb3:27017",
            "priority": 1
        }
    ]
}

rs.initiate(rsconf);
rs.conf();
```

3) Check the configuration with the following command:

```
rs.status()
```

4) Shut down from a new shell the primary node. To shut down a node use the following command

```
docker-compose stop name_primary_mongo
```

where *name_docker_mongo* is the name of the mongo container running the primary node.

5) Identify the new primary node
6) Restart the shutted down node

```
docker-compose restart name_primary_mongo
```

7) Re-check the status of replica set

Query solutions and expected outputs

1) **db.restaurants.find({"cost":"medium"}).pretty()**

```
{
	"_id" : ObjectId("5fa6598843092fb2d4cae545"),
	"name" : "ToorSeafoodrestaurant",
	"tag" : [
		"seafood",
		"expensive"
	],
	"orderNeeded" : false,
	"maxPeople" : 100,
	"review" : 4.3,
	"cost" : "medium",
	"location" : {
		"type" : "Point",
		"coordinates" : [
			45.0664,
			7.6609
		]
	},
	"contact" : {
		"phone" : "+390223456245",
		"facebook" : "ToorSeaFood"
	}
}
{
	"_id" : ObjectId("5fa6598843092fb2d4cae54d"),
	"name" : "Mcdownloads",
	"tag" : [
		"fastfood"
	],
	"orderNeeded" : false,
	"maxPeople" : 70,
	"review" : 3.9,
	"cost" : "medium",
	"location" : {
		"type" : "Point",
		"coordinates" : [
			45.0696,
			7.6503
		]
	},
	"contact" : {
		"Facebook" : "McdownloadTorino",
		"website" : "mcdownloads.com"
	}
}
{
	"_id" : ObjectId("5fa6598843092fb2d4cae54e"),
	"name" : "OldNavyHamburgar",
	"tag" : [
		"hamburger",
		"fastfood"
	],
	"orderNeeded" : false,
	"maxPeople" : 100,
	"review" : 4.5,
	"cost" : "medium",
	"location" : {
		"type" : "Point",
		"coordinates" : [
			45.0772,
			7.674
		]
	},
```

```
        "contact" : {
                "phone" : "+396763452345",
                "facebook" : "OldNavyHamburgar"
        }
}
```

2) **db.restaurants.find({review:{$gt:4},$or:[{cost:"medium"},{cost:"low"}]}, {name:1, maxPeople:1})**

```
{ "_id" : ObjectId("5fa6598843092fb2d4cae545"), "name" : "ToorSeafoodrestaurant",
"maxPeople" : 100 }
{ "_id" : ObjectId("5fa6598843092fb2d4cae546"), "name" : "PandaParadise",
"maxPeople" : 50 }
{ "_id" : ObjectId("5fa6598843092fb2d4cae54a"), "name" : "IlDivinPanino",
"maxPeople" : 10 }
{ "_id" : ObjectId("5fa6598843092fb2d4cae54c"), "name" : "Smartbar", "maxPeople"
: 15 }
{ "_id" : ObjectId("5fa6598843092fb2d4cae54e"), "name" : "OldNavyHamburgar",
"maxPeople" : 100 }
```

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}}, {name:1, 'contact.phone':1, _id:0})**

```
{ "name" : "Takitai", "contact" : { "phone" : "+39087673456" } }
{ "name" : "IlDivinPanino", "contact" : { "phone" : "+393319416860" } }
{ "name" : "IlTempo", "contact" : { "phone" : "+398772376563" } }
```

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.createIndex({location: "2dsphere"})**

db.restaurants.findOne({location:{$near:{$geometry:{type: "Point",coordinates: [45.0644,7.6598]}}}})

```
{
        "_id" : ObjectId("5fa6598843092fb2d4cae54a"),
        "name" : "IlDivinPanino",
        "tag" : [
                "casual",
                "goodforkids"
        ],
        "orderNeeded" : false,
```

```
        "maxPeople" : 10,
        "review" : 4.6,
        "cost" : "low",
        "location" : {
                "type" : "Point",
                "coordinates" : [
                        45.0645,
                        7.6608
                ]
        },
        "contact" : {
                "phone" : "+393319416860",
                "website" : "ildivinpanino.it"
        }
}
```

7) db.restaurants.createIndex({location: "2dsphere"})
db.restaurants.find({location:{$near:{$geometry:{type: "Point",coordinates: [45.0623,7.6627]},$maxDistance: 500}}}).count()

```
3
```

8) db.restaurants.updateMany({tag:'italian'},{$addToSet: {tag:'pizza'}})

```
{ "acknowledged" : true, "matchedCount" : 2, "modifiedCount" : 1 }
```

9) db.restaurants.updateMany({tag:'fastfood'},{$inc: {review:-0.2}})

```
{ "acknowledged" : true, "matchedCount" : 2, "modifiedCount" : 2 }
```

10) db.restaurants.aggregate([{$match:{review:{$gt:3}}}, {$unwind: "$tag"}, {$group:{_id:"$tag",count: { $sum: 1 } } },
{$match:{count:{$gte:2}}}])

```
{ "_id" : "pizza", "count" : 2 }
{ "_id" : "italian", "count" : 2 }
{ "_id" : "fastfood", "count" : 2 }
{ "_id" : "japanese", "count" : 2 }
```

11) db.restaurants.aggregate([{$group:{ _id:"$cost", review_max:{$max: "$review" }, review_min:{$min: "$review" }, review_avg:{$avg: "$review"} , count:{$sum:1} }}, {$sort: {count:-1} }])

```
{ "_id" : "low", "review_max" : 4.7, "review_min" : 3.8, "review_avg" : 4.3,
"count" : 5 }
{ "_id" : "medium", "review_max" : 4.3, "review_min" : 3.6999999999999997,
"review_avg" : 4.1, "count" : 3 }
{ "_id" : "high", "review_max" : 4.2, "review_min" : 4.2, "review_avg" : 4.2,
"count" : 2 }
```

12) db.restaurants.aggregate([{$sort: {maxPeople:1}}, {'$group': {'_id': null, 'value': {'$push': '$maxPeople'}}}, {'$project': {'_id': 1,'50': {'$arrayElemAt': ['$value', {'$floor': {'$multiply': [0.50, {'$size': '$value'}]}}]} }}])

```
{ "_id" : null, "50" : 75 }
```