Data Management and Visualization

Politecnico di Torino

NoSQL in MongoDB – Practice 6

Part 1 - Compass

The practice purpose is to become familiar with MongoDB Compass tool. In this practice you are required to explore data and write some queries to retrieve data from a NoSQL database based on MongoDB.

1) Setup and remote database connection

MongoDB Compass Install (Windows/Linux)

Download MongoDB Compass using one of the following links:

- Ubuntu (.deb): https://downloads.mongodb.com/compass/mongodb-compass_1.18.0_amd64.deb • RedHat (.rpm): https://downloads.mongodb.com/compass/mongodb-compass-1.18.0.x86_64.rpm
- Windows (.exe) 64 bit: https://downloads.mongodb.com/compass/mongodb-compass-1.18.0-win32-x64.exe
- Mac OS (.dmg): https://downloads.mongodb.com/compass/mongodb-compass-1.18.0-darwin-x64.dmg

Install and open the application

Connection Setup

1. Connect to the remote database using the following connection parameters:
    a. **Hostname:** bigdatadb.polito.it
    b. **Port:** 27017
    c. **Authentication:** Username/Password
    d. **Username:** Compass
    e. **Password:** Compass19!
    f. **Authentication database:** dbdmg
    g. **SSL:** Unvalidated (insecure)
2. (Optional) Specify a **Favourite Name** to easily connect to the database in the future.
3. Click on **Connect.**
4. Access to **dbdmg**
5. Access to a specific partition (**Parkings/Bookings**).

2) Problem specifications

The database contains Car Sharing information divided into two main collections: Bookings and Parkings. The most relevant information for each collection is shown in Table 1 (Parkings) and 2 (Bookings).

Name	Type	Description
_id	objectid	Document identifier.
address	string	Parking address of the vehicle.
city	string	City location of the vehicle.
engineType	string	Identifier of the engine type of the vehicle.
exterior	string	String describing the external condition of the vehicle during the parking.
final_date	date	Date and hour of the end of the parking period.
fuel	int32	Fuel level (0-100) during the parking period.
init_date	date	Date and hour of the beginning of the parking period.
interior	string	String describing the internal condition of the vehicle during the parking.
loc	coordinates	Coordinate of the parking location.
plate	int32	Identifier of the vehicle's plate.
smartphoneRequired	boolean	Boolean value denoting if the smartphone is required to start/finish the parking.
vendor	string	Company owner of the vehicle.
vin	string	Identifier of the chassis of the vehicle.

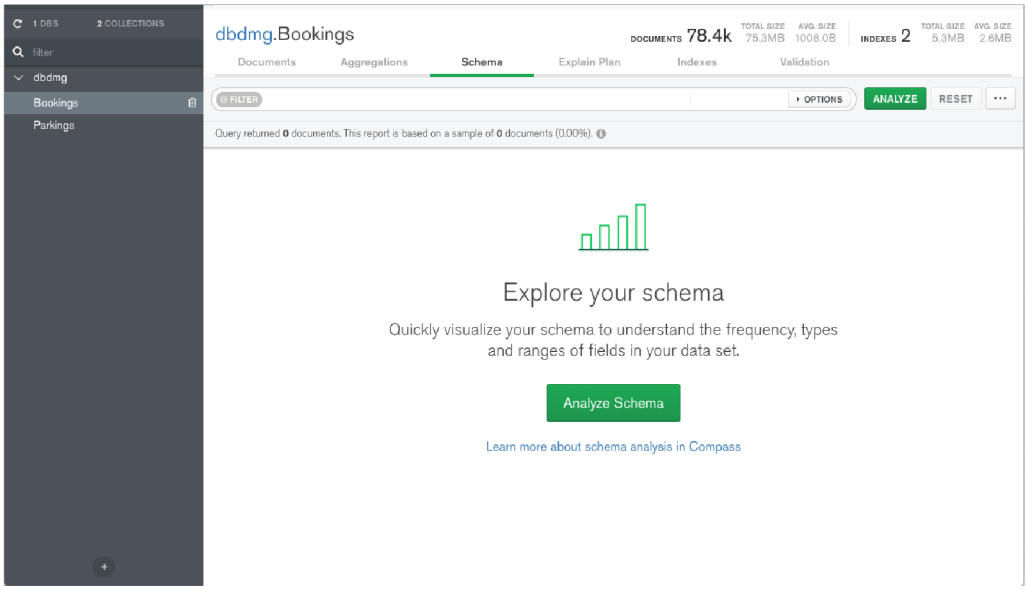
Table 1: **Parkings** database info.

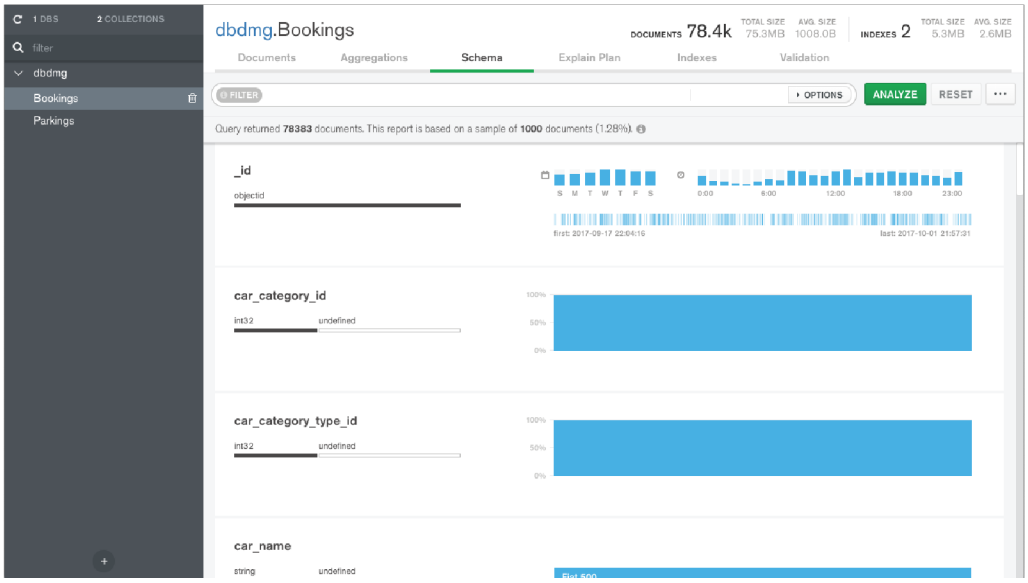
Name	Type	Description
_id	objectid	Document identifier.
car_name	string	Vehicle's model
city	string	City location where the vehicle has been booked.

distance	int32	Distance covered during the vehicle renting.		
driving	Array	**distance**	int32	Distance covered during the vehicle renting (in meters).
		duration	int32	Duration of the renting (in seconds)
engineType	string	Identifier of the engine type of the vehicle.		
exterior	string	String describing the external condition of the vehicle during the renting.		
final_address	string	Address of the final position of the renting period.		
final_date	date	Date and hour of the end of the renting period.		
final_fuel	int32	Fuel level (0-100) at the end of the renting period.		
init_address	int32	Address of the starting position of the renting period.		
init_date	date	Date and hour of the beginning of the renting period.		
init_fuel	int32	Fuel level (0-100) at the beginning of the renting period.		
interior	string	String describing the internal condition of the vehicle during the renting.		
plate	int32	Identifier of the vehicle's plate.		
smartphoneRequired	boolean	Boolean value denoting if the smartphone is required to start/finish the parking.		
vendor	string	Company owner of the vehicle.		
walking	Array	**distance**	int32	Walk distance to reach the vehicle (in meters).
		duration	int32	Duration of the walking trip to reach the vehicle (in seconds).

Table 2: **Bookings** database info.

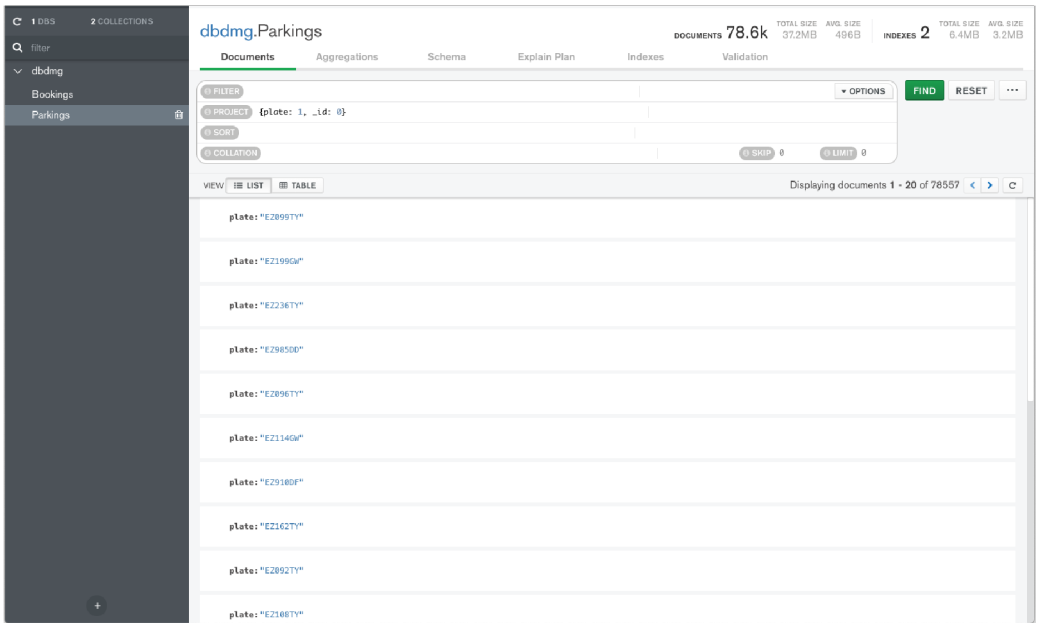
3) Analyze the database using the Schema analyzer





1. ((Bookings) Identify the most common percentage(s) of fuel level at the beginning of the renting period.
2. (Bookings) Identify the most common percentage(s) of fuel level at the end of the renting period.
3. (Parkings) Identify the time range(s) with most parking requests (start parking).
4. (Parkings) Identify the time range(s) with most booking requests (end parking).
5. (Parkings) Visualize on the map the vehicles having the fuel level lower than 5%.

4) Querying the database



1. (Parkings) Find the plates and the parking addresses of the vehicles that begin the booking (end parking) after 2017-09-30 at 6AM.
   (Hint: it is possible to use the function Date("<YYYY-mm-ddTHH:MM:ss>"))
2. (Parkings) Find the addresses and the level of fuel of the vehicles that during the parking period had at least 70% of fuel level. Order the results according to descending value of fuel level.
3. (Parkings) Find the plate, the engine type and fuel level for 'car2go' vehicles (vendor) with good internal and external conditions.
4. (Bookings) For the renting that required a walking distance greater than 15 Km (to reach the vehicle), find the hour and the fuel level at the beginning of the renting period. Order results according to decreasing initial fuel level.

5) Data Aggregation

5. (Bookings) Group documents according to their fuel level at the end of the renting. For each group, select the average fuel level at the beginning of the renting period.
6. (Bookings) Select the average driving distance for each vendor. On average, for which vendor the users cover longer distances?

Part 2 – MongoDB

The objective of the second part 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. MongoDB is already installed at LABINF.

1)Practise Setup

LABINF

a. Create a local folder (e.g.: C:\Users\<S123456>\Desktop\mongo_database) and save its path, from now on called: my_database_path. This folder will contain the DB generate with MongoDB.

b. Navigate to C:\Program Files\MongoDB\4.0\bin and open a command shell in the location (maiusc + right-click -> open command window here).
(E.g. cd C:\Program Files\MongoDB\4.0\bin).

c. Run the following command:
mongod –-dbpath my_database_path

Practice at home - MongoDB community Edition

To on your PC, you need to install **MongoDB Server**.

You can install MongoDB by following the official guide for your operating system.

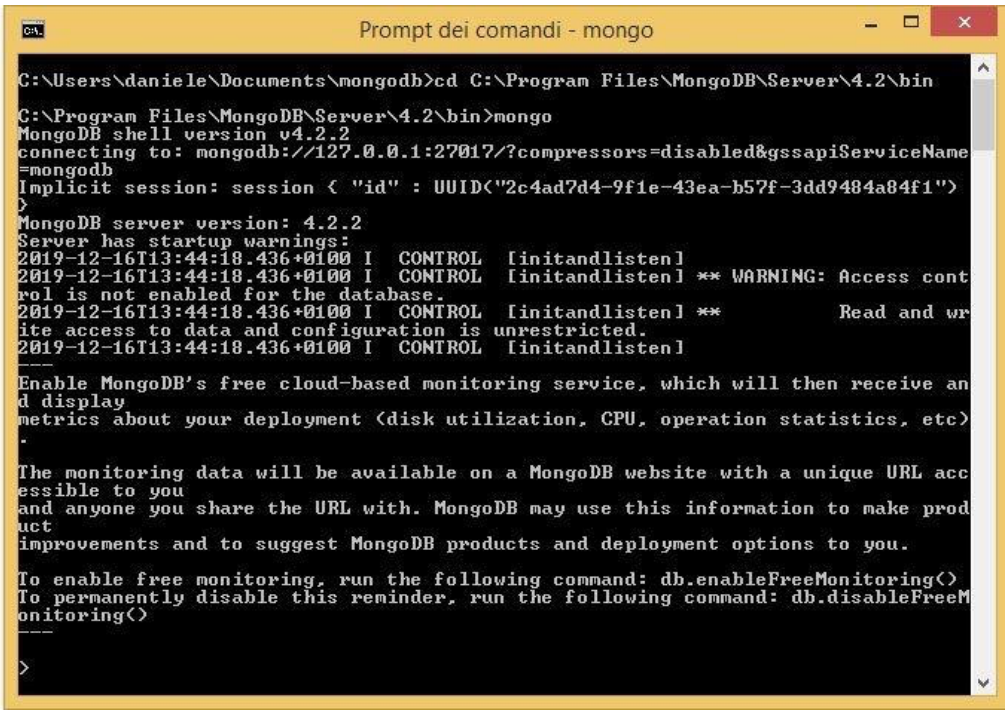
- Linux
- Mac OS
- Windows

For installation on Mac OS, you can follow the official guide (which makes use of Homebrew).

Verify the installation

For the next steps, you need to know how to run `mongod` commands from the terminal. For Linux and Mac OS the commands should be directly available (the executables are loaded into a directory in $ PATH). For Windows, you will need to use the full path (e.g. "C:\Program Files\MongoDB\Server\4.4\bin\mongod.exe").

Output example on Windows:



DB creation

To create a db, it is necessary to execute mongod specifying the parameter --dbpath, that is the path on filesytem where we want to create our database.

mongod –-dbpath my_database_path

To create the db in the directory my_database_path

Ubuntu - If you have an error:

```
systemctl unmask mongod
```

2) Creating the database collection (Windows/Linux)

a. Download the **Restaurants** database in json format from the course website
   E.g. ”C:\Documents\lab\MongoDB\restaurants_collection.json”

b. Open another Command Shell in the folder of MongoDB (previous location)

c. Run the following command:
   mongoimport –-db=restaurantsDB –-collection=restaurants --file=”C:\Documents\lab\MongoDB\restaurants_collection.json” --jsonArray (Modify the json path based on your own configuration)

```
C:\Program Files\MongoDB\Server\4.2\bin>mongoimport --db=restaurantsDB --collection=restaurants --file="E:\DS-DBTech 2018-2019\lab\MongoDB\restaurants_collection.json" --jsonArray
2019-12-18T12:35:11.502+0100    connected to: mongodb://localhost/
2019-12-18T12:35:11.563+0100    10 document(s) imported successfully. 0 document(s) failed to import.
```

Alternative:

mongo use restaurantsDB
db.restaurants.insertMany(<file content>)

d. Run the following command: mongo
You are now logged into the Mongo Shell.

e. Activate the restaurants db:
use restaurantsDB

```
> use restaurantsDB
switched to db restaurantsDB
>
```

f. In order to check the success of the import, run the command:
db.restaurants.find().pretty()

```
> use restaurantsDB
switched to db restaurantsDB
> db.restaurants.find().pretty()
{
        "_id" : "002",
        "name" : "PandaParadise",
        "tag" : [
                "chinese",
                "japanese"
        ],
        "orderNeeded" : false,
        "maxPeople" : 50,
        "review" : 4.7,
        "cost" : "low",
        "location" : {
                "type" : "Point",
                "coordinates" : [
                        45.0671,
                        7.6627
                ]
        },
        "contact" : {
                "phone" : "+395487634998",
                "facebook" : "PandaP"
        }
}
{
        "_id" : "001",
```

3) Query on Restaurants database

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 }
}
```

Running queries of interest:

a. Find all restaurants whose cost is medium
b. Find all restaurants whose review is bigger than 4 and cost is medium or low

c. Find all restaurants that can contain more than 5 people and:
    i. whose tag contains "italian" or "japanese" and cost is medium or high
    OR
    ii. whose tag does not contain neither "italian" nor "japanese", and whose review is higher than 4.5
d. Calculate the average review of all restaurants
e. Count the number of restaurants whose review is higher than 4.5 and can contain more than 5 people
f. Run query n. d) using the Map-Reduce paradigm
g. Run query n. e) using the Map-Reduce paradigm
h. Find the restaurant in the collection which is nearest to the point [45.0644, 7.6598] Hint: remember to create the geospatial index.
i. Find how many restaurants in the collection are within 500 meters from the point [45.0623, 7.6627]