Data Science and Database Technology

Politecnico di Torino

NoSQL on 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

Install MongoDB Compass

1. Download MongoDB Compass a https://www.mongodb.com/try/download/compass , Install and Open it
2. Clink on "Create a free cluster" on the right.
    a. make an account on MongoDB
    b. Select "M0" (free cluster). Keep the default settings about the Provider and the Region.
    c. Choose a name and Create the cluster.
    d. Go to "Database Access" on the left-side menu
        i. Click on "Add new database user"
        ii. Select the authentication method with password
        iii. Fill the Password Authentication module
        iv. Select "Atlas admin" as Built-in Role
        v. Keep all the other default settings
    e. configure remote access
        i. click on "Network access"
        ii. click on "add IP address"
        iii. insert 0.0.0.0/0
3. Get the connection string from the MongoDB Atlas server
    a. Go on "Database" page from the left-side menu
    b. Click on "Connect" next to cluster you have created
    c. select "Compass"
    d. copy the connection string
4. Paste the connection string in MongoDB Compass
5. Click on Connect

Creation database

1. Click on Create a new Database by clicking on "+" next to Databases on the left.
    a. Assign a name
    b. Insert the name "Booking" in the field "Collection Name"
    c. Keep all the other default settings
2. Add a new collection by clicking on the "+" next to the database you have created in the previous step
    a. Click on "create collection".
    b. Add name "Parkings"
    c. Keep all the other default settings
3. download data from the dbdmg website

4. Add data to each collection
    a. Select a collection
    b. Click on "Import data"
    c. Select the corresponding collection
    d. Select Json type
    e. Click on import
5. Select the Parking collection
6. Go to "Indexes" board
7. Crate a index 2dsphere on the loc field

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).

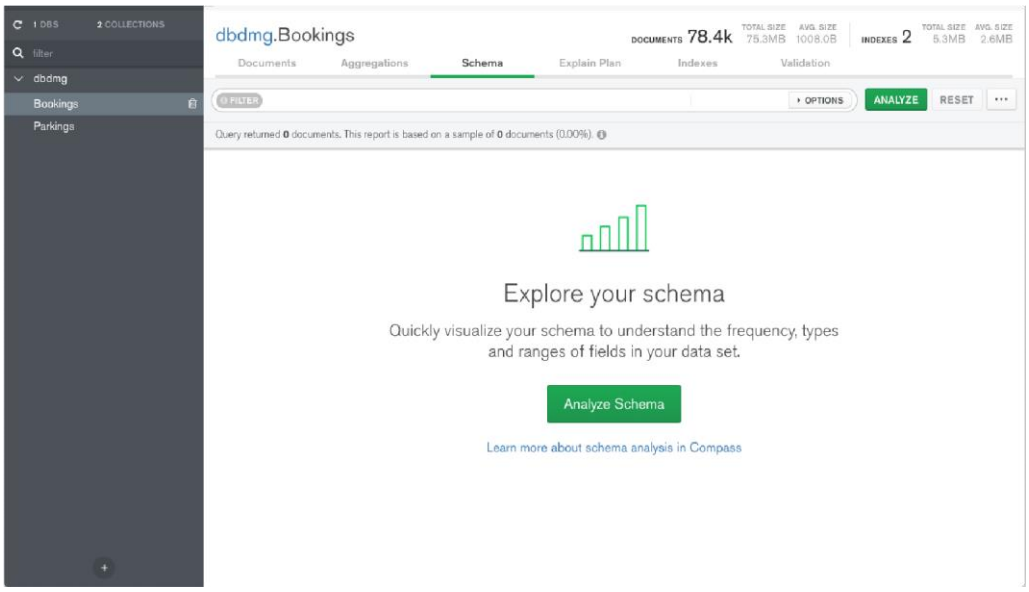
Name	Type	Description
_id	objectid	Document identifier.
address	string	Parking address of the vehicle.
city	string	City location of the vehicle.
engineType	string	Identifier of the engine type of the vehicle.
exterior	string	String describing the external condition of the vehicle during the parking.
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.

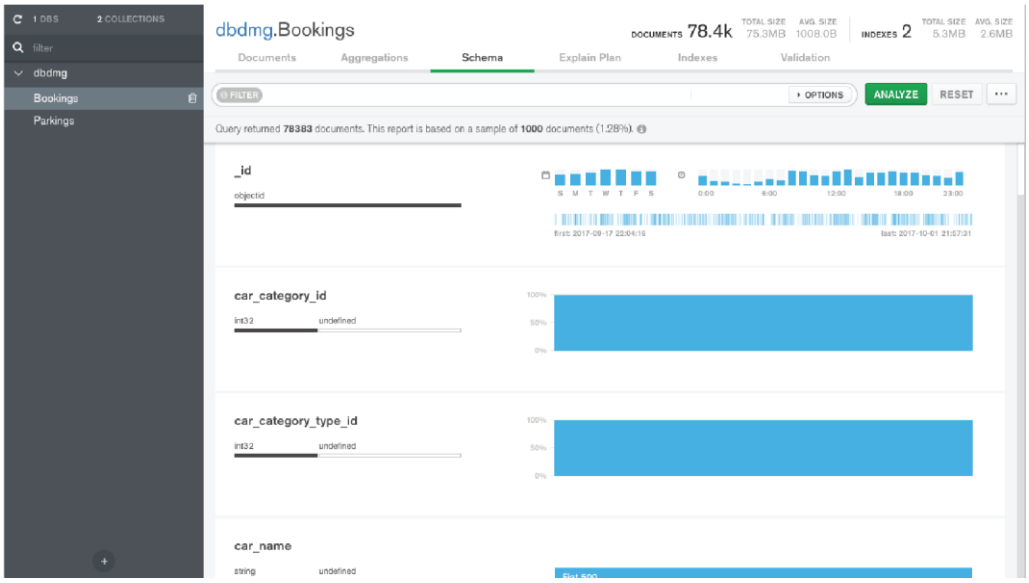
Table 1: **Parkings** database info.

_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.

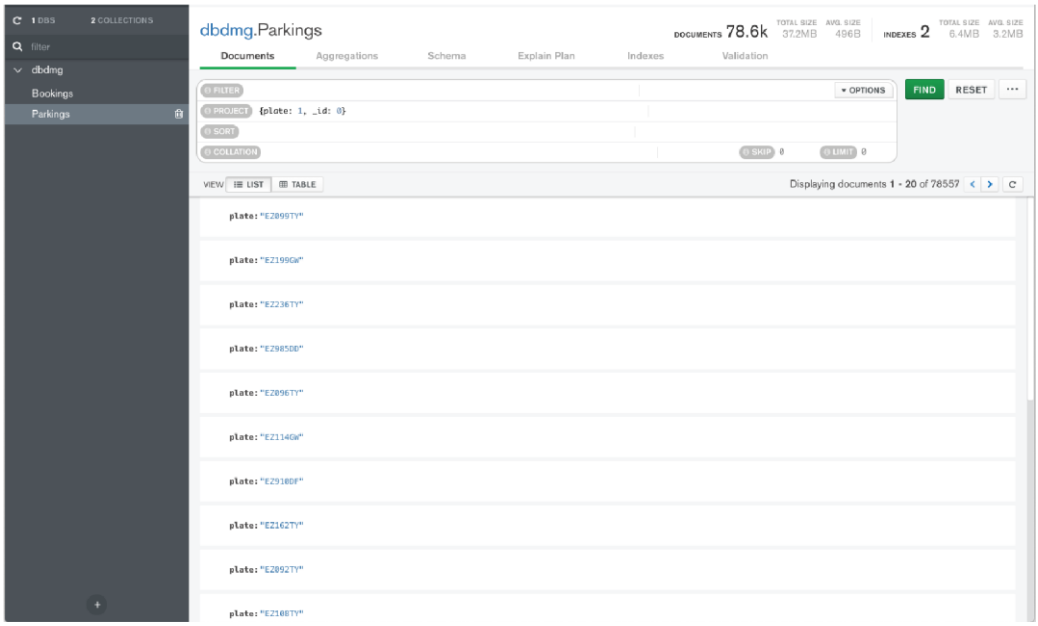
3) Analyze the database using the Schema analyzer





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: new 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 Shell

The objective of the second part of the practice is to execute queries on **MongoDB Shell**. You can download the software at the following link: https://www.mongodb.com/try/download/shell

1)Import database

a. Download the Restaurants database in Json from the dbdmg website and upload it as you have already done on MongoDB Compass
b. Access the database following the instructions of Atlas by clicking Connect →Compass (not Shell)
c. Activate the Restaurant database with the following command:

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

d. Check that the db has been uploaded correctly

```
> 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",
```

2) 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 usersorderNeeded: <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]