

Lab06

martedì 27 dicembre 2022 11:46



Lab6_Mon...

Parte 1 - Compass

L'obiettivo di questa prima parte del laboratorio è quello di utilizzare **MongoDB Compass** come tool visuale per lavorare con un db non relazionale. In questo esercizio vi verrà richiesto di analizzare i dati e scrivere alcune query per prelevare informazioni da un database NoSQL basato su MongoDB.

1) Setup e connessione al database remoto

È necessario installare MongoDB Compass sul proprio computer.

Installazione di MongoDB Compass (Windows/Linux)

Scaricare ed installare MongoDB Compass da uno dei link seguenti (oppure, direttamente dalla [pagina di download](#) per l'ultima versione).

- 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>

Setup della connessione

Aprire MongoDB Compass e connettersi al database remoto creato per questa esercitazione con i seguenti step:

1. Connetersi al database remoto, utilizzando i seguenti parametri:
 - 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. (Opzionale) Specificare “**Favourite Name**” per utilizzare la connessione in futuro. Click su **Save Favourite**.
3. Click su **Connect**.
4. Accedere a **dbdmg** facendo click sul link mostrato dall'interfaccia grafica.
5. Accedere ad una delle due collezioni presenti nel database (**Parkings/Bookings**).

2) Specifiche del problema

Il database contiene dati relativi al tema Car Sharing divisi in due collezioni principali: Bookings e Parkings. Le informazioni più rilevanti per ogni collezione sono mostrate in Tabella 1 (Parkings) e 2 (Bookings).

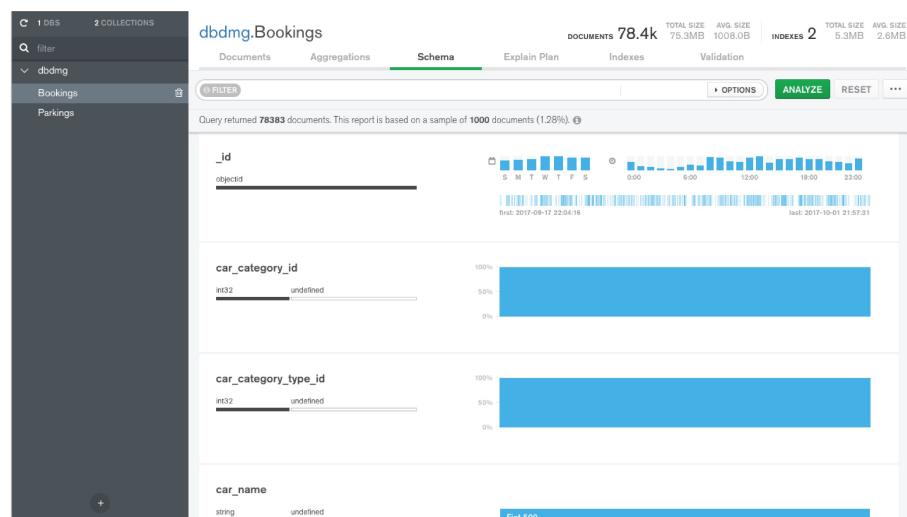
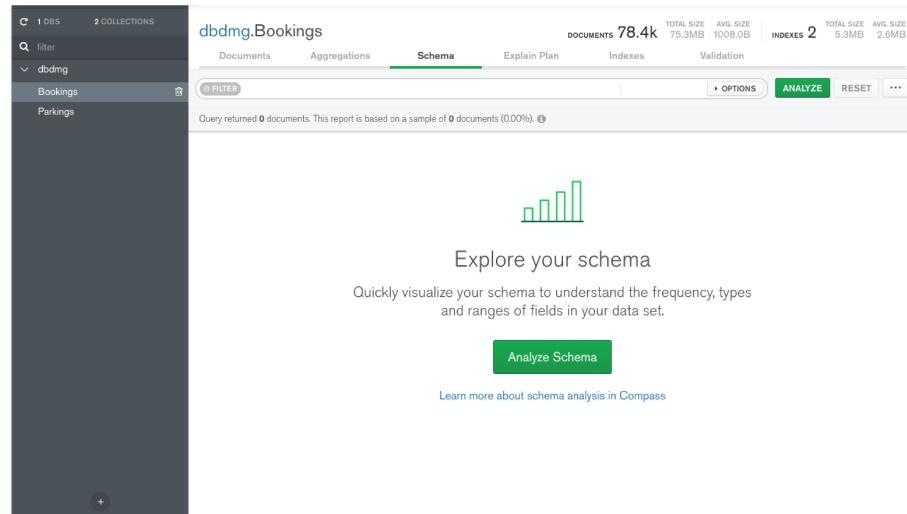
Name	Type	Description
_id	objectid	Identificatore del documento.
address	string	Indirizzo di parcheggio del veicolo.
city	string	Città in cui si trova il veicolo.
engineType	string	Identificatore del tipo di motore del veicolo.
exterior	string	Stringa contenente le condizioni esterne del veicolo durante il parcheggio.
final_date	date	Data e ora di fine del periodo di parcheggio.
fuel	int32	Livello di carburante (0-100) durante il parcheggio.
init_date	date	Data e ora di inizio del periodo di parcheggio.
interior	string	Stringa contenente le condizioni interne del veicolo durante il parcheggio.
loc	coordinates	Coordinate del luogo di parcheggio.
plate	int32	Identificatore della targa del veicolo.
smartphoneRequired	Boolean	Valore booleano che specifica se e' necessario uno smartphone per iniziare/finire il parcheggio.
vendor	string	Azienda che mette a disposizione i veicoli.
vin	string	Identificatore telaio del veicolo.

Tabella 1: **Parkings**.

Name	Type	Description		
_id	objectid	Identificatore documento.		
car_name	string	Modello del veicolo.		
city	string	Città in cui il veicolo è stato prenotato.		
driving	object	distance	int32	Distanza coperta dal veicolo durante il noleggio (metri).
		duration	int32	Durata del noleggio (in secondi)
engineType	string	Identificatore del tipo di motore del veicolo.		
exterior	string	Stringa che descrive le condizioni esterne del veicolo durante il noleggio.		
final_address	string	Indirizzo della posizione finale del periodo di noleggio.		
final_date	date	Data e ora di fine noleggio.		
final_fuel	int32	Livello di carburante (0-100) alla fine del noleggio.		
init_address	int32	Indirizzo della posizione iniziale del periodo di noleggio.		
init_date	date	Data e ora di inizio noleggio.		
init_fuel	int32	Livello di carburante (0-100) all'inizio del noleggio.		
interior	string	Stringa che descrive le condizioni interne del veicolo durante il noleggio.		
plate	int32	Identificatore della targa del veicolo.		
smartphoneRequired	Boolean	Valore booleano che specifica se è necessario uno smartphone per iniziare/finire il noleggio.		
vendor	string	Azienda che mette a disposizione i veicoli.		
walking	object	distance	int32	Distanza per raggiungere il veicolo a piedi (metri).
		duration	int32	Durata del cammino per raggiungere il veicolo (secondi).

Tabella 2: **Bookings**.

3) Analizzare il database con il tool “Schema analyzer”



1. (Bookings) Identificare i valori più comuni relativi al livello di carburante all'inizio del noleggio.
2. (Bookings) Identificare i valori più comuni del livello di carburante alla fine del noleggio.
3. (Parkings) Identificare gli intervalli di tempo con più richieste di parcheggio (init_date di Parkings).
4. (Parkings) Identificare gli intervalli di tempo con più richieste di noleggio (final_date di Parkings).
5. (Parkings) Visualizzare sulla mappa i veicoli con livello di carburante minore del 5%.

4) Eseguire query sul database

The screenshot shows the db4o studio interface. On the left, there's a sidebar with '1 DBS' and '2 COLLECTIONS' (Bookings, Parkings). The main area is titled 'dbdmg.Parkings' and shows a list of documents. The 'plate' field is highlighted in green. The search bar at the bottom contains the query 'plate: "EZ099TY"'.

1. (Parkings) Trovare le targhe (plate) e gli indirizzi di parcheggio dei veicoli che cominciano il noleggio (final date) dopo il 30-09-2017 alle 6 di mattina. (Consiglio: è possibile usare la funzione Date("<YYYY - mm - ddTHH: MM: ssZ>"), dove Z indica il fuso orario UTC)
2. (Parkings) Trovare gli indirizzi ed il livello di carburante dei veicoli che durante il periodo di parcheggio hanno avuto almeno il 70% di livello di carburante. Ordinare in modo decrescente i risultati per livello di carburante.
3. (Parkings) Trovare la targa, il tipo di motore ed il livello di carburante per i veicoli 'car2go' (vendor) con buone condizioni interne ed esterne.
4. (Bookings) Per i noleggi che necessitano di una distanza a piedi maggiore di 15 Km (per raggiungere il veicolo), trovare l'ora ed il livello di carburante all'inizio del noleggio. Ordinare i risultati secondo livello di carburante iniziale decrescente.

5) Data Aggregation

5. (Bookings) Raggruppare i documenti in base al livello di carburante alla fine del noleggio. Per ogni gruppo selezionare il livello di carburante medio all'inizio del periodo di noleggio.
6. (Bookings) Selezionare la distanza media coperta dai veicoli di ogni azienda (vendor). In media, per quali aziende gli utenti coprono le maggiori distanze?

Parte 2 – MongoDB su shell

La seconda parte dell'esercitazione consiste nell'eseguire query in locale su una base dati non relazionale mediante la shell di MongoDB. Per eseguire l'esercitazione sul proprio pc è necessario installare **MongoDB Server** come indicato dal [tutorial ufficiale](#).

1) Setup dell'esercitazione

- a. Creare una cartella (e.g.: `/data/db`), da ora in poi chiamata: *my_database_path*.
Questa cartella conterrà i database generati con MongoDB.
- b. **Su Windows:** posizionarsi nella cartella di installazione di MongoDB mediante Prompt dei Comandi (o shell Linux). Ad esempio:
`cd C:\Program Files\MongoDB\4.0\bin`
- c. Eseguire il seguente comando per specificare la cartella di salvataggio dei DB:
`mongod --dbpath my_database_path`

2) Importare il database Restaurants (Windows/Linux)

- a. Scaricare il database **Restaurants** in formato json dal sito del corso (sezione Laboratori). Posizionare il file in una cartella a piacere.
- b. Aprire un nuovo terminale e (**su Windows**) posizionarsi nella cartella di installazione di MongoDB mediante Prompt dei Comandi:
`cd C:\Program Files\MongoDB\Server\4.0\bin`
- c. Eseguire il seguente comando per importare il database:
`mongoimport --db=restaurantsDB --collection=restaurants --file=path_to_restaurants_collection.json --jsonArray`
(Modificare il percorso del file json in base alla propria configurazione)

```
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.
```

- d. Eseguire da terminale il seguente comando per aprire la shell di MongoDB Server:
`mongo`
 - e. Attivare il database restaurants digitando il seguente comando:
`use restaurantsDB`
- ```
> use restaurantsDB
switched to db restaurantsDB
>
```
- f. Controllare che il database sia stato caricato correttamente:

```
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 sul database Restaurants

- a. Trovare tutti i ristoranti con costo “medium”
- b. Trovare tutti i ristoranti il cui valore di review è maggiore di 4 ed il costo è “medium” oppure “low”
- c. Trovare tutti i ristoranti che possono ospitare più di 5 persone (maxPeople) e:
  - i. hanno un tag che contiene “italian” oppure “japanese” e hanno costo “medium” oppure “high”  
**OPPURE:**
  - ii. hanno un tag che non contiene né “italian” né “japanese” e hanno review maggiore di 4.5
- d. Calcolare il valore di review medio di tutti i ristoranti
- e. Contare il numero di ristoranti il cui valore di review è maggiore di 4.5 e che possono ospitare più di 5 persone
- f. Eseguire la query d) usando il paradigma Map-Reduce
- g. Eseguire la query e) usando il paradigma Map-Reduce
- h. Trovare il ristorante più vicino al punto [45.0644, 7.6598]  
Consiglio: ricordarsi di creare un “geospatial index”.
- i. Trovare il numero di ristoranti che sono entro 500 metri dal punto [45.0623, 7.6627]

A) db.restaurants.find({cost:"medium"})  
B) db.restaurants.find( { review : {\$gt : 4 }, \$or: [ {cost : "medium"}, {cost: "low"} ]})  
C)  
i. db.restaurants.find({\$and :[ {maxPeople : {\$gt : 5}}, { \$or:[{tag : 'italian'}, {tag : 'japanese'}]}],{ \$or:[{cost: 'medium'},{cost: 'high'}]} })  
ii. db.restaurants.find({\$and :[ {maxPeople : {\$gt : 5}}, { tag : {\$nin: ['italian', 'japanese']}}, { review : {\$gt : 4.5 }}}]})  
D) db.restaurants.aggregate( [ { \$group: { \_id: null, avg: { \$avg: "\$review" } }}])  
E) db.restaurants.find({\$and:[{review : {\$gt : 4.5}}, {maxPeople : {\$gt : 5}}]}).count()  
F) db.restaurants.mapReduce( function(){emit(null, this.review);}, function(key,values){return Array.avg(values);}, {out:{inline:1}})  
G) db.restaurants.mapReduce( function(){emit(null, 1);}, function(key,values){var sum=0; for(var i in values) sum+=values[i]; return sum;}, { query: {\$and:[{review : {\$gt : 4.5}}, {maxPeople : {\$gt : 5}}]}}, out:{inline:1}})  
H) db.restaurants.createIndex({location: "2dsphere"})  
db.restaurants.find({location : {\$near: {\$geometry : { type:"Point", coordinates:[45.0644, 7.6598]} }}}}.limit(1)  
I) db.restaurants.find({location : {\$near: {\$geometry : { type:"Point", coordinates:[45.0644, 7.6598]}}, \$maxDistance:500 }})}