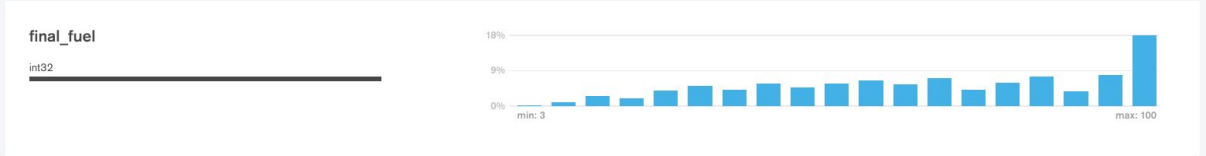
# MongoDB

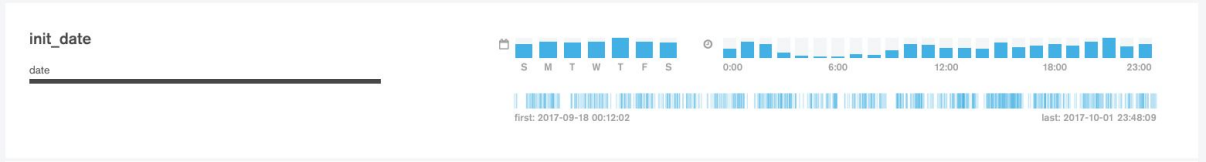
MongoDB COMPASS Schema analyser

1. (Bookings) Identificare i valori più comuni relativi al livello di carburante all'inizio del noleggio. 95-100

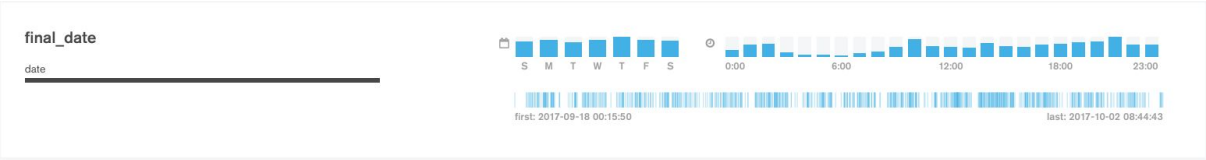

2. (Bookings) Identificare i valori più comuni del livello di carburante alla fine del noleggio. 95-100




3. (Parkings) Identificare gli intervalli di tempo con più richieste di parcheggio (init_date di Parkings).
2017-28-09
20,21,1,2
giovedì



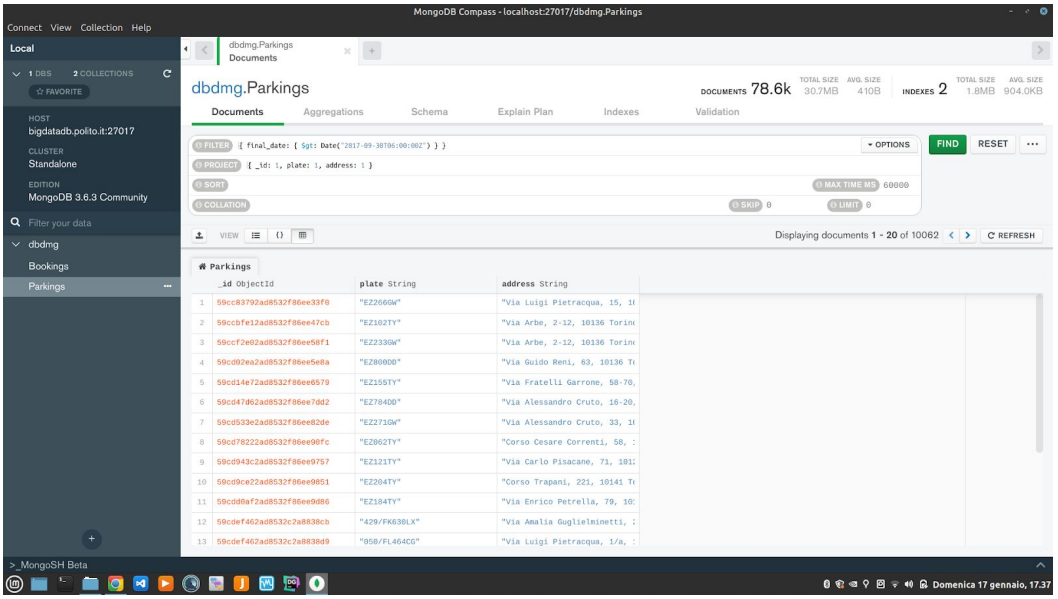

4. (Parkings) Identificare gli intervalli di tempo con più richieste di noleggio (final_date di Parkings).



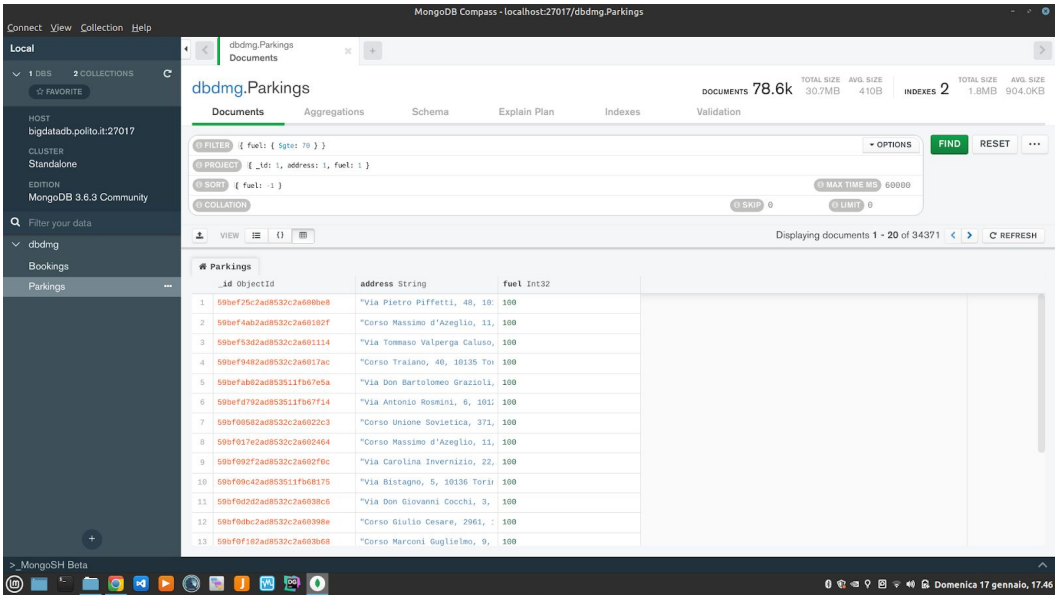

5. (Parkings) Visualizzare sulla mappa i veicoli con livello di carburante minore del 5%.
Visualizzare mappa?

QUERY

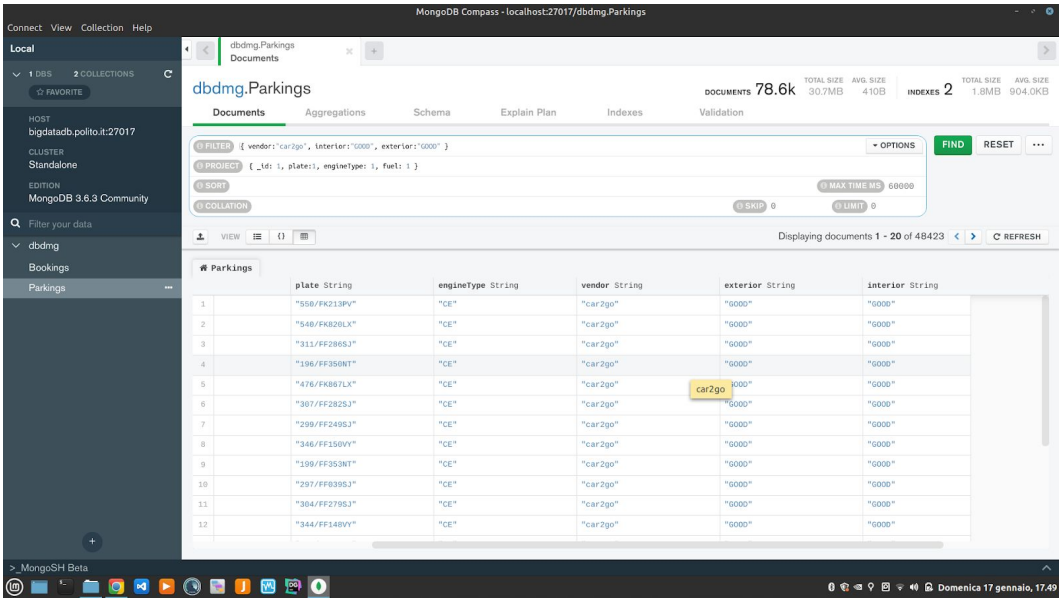
1. (Parkings) Trovare le targhe (plate) e gli indirizzi di parcheggio dei veicoli che cominciano il noleggio (final date) dopo il 30-09-2017 alle 6 di mattina. (Consiglio: è possibile usare la funzione Date("YYYY-mm-ddTHH:MM:ssZ"), dove Z indica il fuso orario UTC )



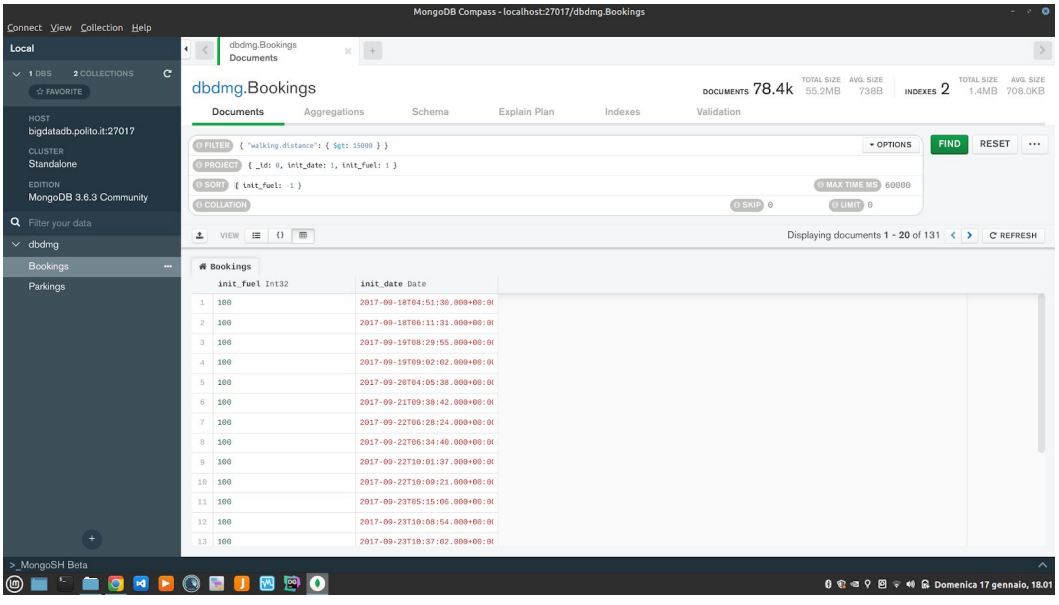

2. (Parkings) Trovare gli indirizzi ed il livello di carburante dei veicoli che durante il periodo di parcheggio hanno avuto almeno il 70% di livello di carburante. Ordinare in modo decrescente i risultati per livello di carburante.




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.

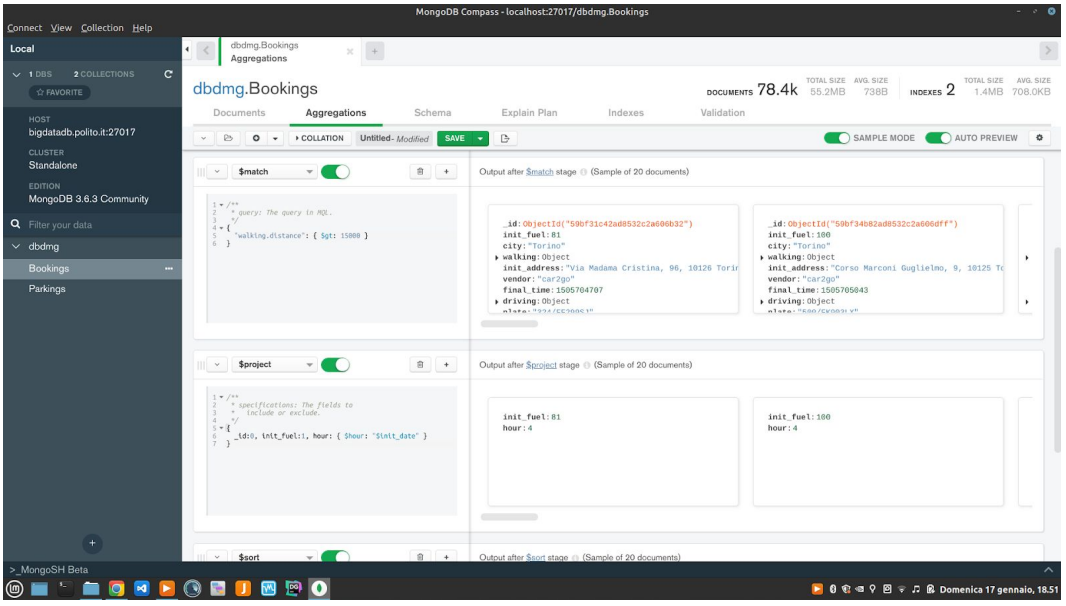


DATA AGGREGATION

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? enjoy


(7. (Bookings) vedi 5.)





SHELL MONGO

a. Trovare tutti i ristoranti con costo "medium"

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

b. Trovare tutti i ristoranti il cui valore di review è maggiore di 4 ed il costo è "medium" oppure "low"

db.restaurants.find( { review: {$gt: 4} , $or: [{ cost: "medium" }, { cost: "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

```
> db.restaurants.find(
...                 {maxPeople: {$gt:5} ,
...                                 $or:[ {tag: {$in: ["italian", "japanese"]} , $or: [ {cost:"medium"},{cost:"high"} ] } ,
...                                     {tag: {$nin:["italian", "japanese"]} , review: {$gt:4.5}                    }   ] } ).pretty()
```

d. Calcolare il valore di review medio di tutti i ristoranti

```
> db.restaurants.aggregate([ {$group: { _id: null, average_review: {$avg: "$review"}} }]).pretty()
{ "_id" : null, "average_review" : 4.26 }
```

e. Contare il numero di ristoranti il cui valore di review è maggiore di 4.5 e che possono ospitare più di 5 persone

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

f. Eseguire la query d) usando il paradigma Map-Reduce

```
> var mapIdtoReview = function() { emit( null, this.review); };
> var mapIdtoReview2 = function(doc) { emit( null, doc.review); };
> var reducetoAvgReview = function(key_null, values_reviews) {
  S=Array.sum(values_reviews);
  N=values_reviews.length;
  AVG=S/N;
  return AVG; };
> db.restaurants.mapReduce( mapIdtoReview, reducetoAvgReview, { out:"output_of_mapReduce" } )
> db.output_of_mapReduce.find()
{ "_id" : null, "value" : 4.26 }
```

g. Eseguire la query e) usando il paradigma Map-Reduce

```
> var mapIdtoId = function() { emit( null, this._id); };
> var reducetoCount = function(key_null, values__id) { return values__id.length; };
> db.restaurants.mapReduce( mapIdtoId, reducetoCount, { out:"output_of_mapReduce", query: { review: {$gt: 4.5} ,
maxPeople: {$gt: 5} } } )
> db.output_of_mapReduce.find()
{ "_id" : null, "value" : 2 }
```

h. Trovare il ristorante più vicino al punto [45.0644, 7.6598] Consiglio: ricordarsi di creare un “geospatial index”.

```
db.restaurants.createIndex( {location:"2dsphere"} )
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "ok" : 1
}
> db.restaurants.find( {location: {$near: {$geometry: {type:"Point", coordinates: [ 45.0644, 7.6598 ] } } } } ).pretty()
```

(ordine di distanza crescente) (va bene anche findOne, ma senza pretty())

i. Trovare il numero di ristoranti che sono entro 500 metri dal punto [45.0623, 7.6627]

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