TP3 – NOSQL MONGODB RESTAURANT INSPECTIONS

*Manon GARDIN*

*Matias OTTENSEN*

*Alexandre GARNIER*

*Tiphaine KACHKACHI*

Create the Database

After running our container for MongoDb we can connect ourselves to mongodbCompass (or in the mongodb CLI)

Une image contenant texte, Police, capture d’écran, logo

Description générée automatiquement

Une image contenant texte, capture d’écran, Police, logo

Description générée automatiquement

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement  
Une image contenant texte, capture d’écran, ligne, Police

Description générée automatiquement

If we were to do it using the Cli of mongodb we could do

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

To connect ourselves to the database

# Import the data

To import our data we can directly use the button :

# Une image contenant texte, capture d’écran, Police, nombre Description générée automatiquement

All objects were added to our database !

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Queries

Since our dataset is a difficult one, we are going to do 8 easy queries, 1 complex and 1 hard.

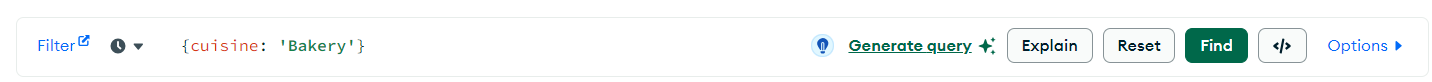
Easy Queries

## Query 1 : To find all Bakery restaurants

The expected output of this query is to get the information of all bakeries and no other resaturants.

In the filter section :

{cuisine: 'Bakery'}



Results :

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

We got 691 results, as expected, we only find bakeries in the output. 

## Query 2 : Find all restaurants that are located on Park Avenue in Brooklyn

All restaurants which appear in the output should then be from there and none of these restaurants should be forgotten.

Une image contenant texte, Police, ligne, capture d’écran

Description générée automatiquement Une image contenant texte, Police, capture d’écran, ligne

Description générée automatiquement

{

"address.street": "Park Avenue",

"borough": "Brooklyn"

}

Results :

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

We had few results, all of them are restaurants on Park Avenue in Brooklyn.

Une image contenant Police, Graphique, texte, blanc

Description générée automatiquement

## Query 3 : To find all restaurants who have been graded at least 6 times

The output should only be composed of resaturants which have been graded 6 times or more.

[{

$match: {

$expr: { $gte: [{ $size: "$grades" }, 6] }

}

}]

The $match is trying to find to find items who will be having the following characteristics.

$size returns the number of elements in an array. Here we look for the numbers of grades in ‘grades’ and we want them to be equal of higher than 6. We do that with $gte which stands for ‘greater than or equal’ operator.

The $expr is used to allow the condition to be used as a filter in the match.

Une image contenant texte, Police, ligne, capture d’écran

Description générée automatiquement

Results (sample displayed) :

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

All restaurants visible in the output have a minimum of 6 grades, as expected.

## Query 4 : List all types of Restaurants and how many of them are they, descending order (aggregation)

The expected output of this query is a list of all restaurants types as well as the number of restaurants available for each.

[

{

$group: {

\_id: "$cuisine",

count: { $sum: 1 }

}

},

{

$sort: { count: -1 }

}

]

Une image contenant texte, Police, capture d’écran, écriture manuscrite

Description générée automatiquement

In order to do so we use $group to group the results by cuisine type (hence the \_id\*: “$cuisine” to use $cuisine as a grouping key).

We count the numbers of restaurants by summing the number of elements of each group ($sum: 1) and store the result in a field count during the grouping operation.

Lastly, we simply order them according to the number of restaurants with $sort (to do so in descending order we input -1).

Results (not everything is on the screen) :

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

The output is expectedly a list of restaurant types as id with their respective number of restaurants.

## Query 5 : Find all restaurants that were graded at least once in 2012

All restaurants in the output should contain at least one grade from 2012 and no other.

{

"grades": {

$elemMatch: {

"date": {

$gte: new Date("2012-01-01"),

$lt: new Date("2013-01-01")

}

}

}

}

Une image contenant texte, Police, ligne, nombre

Description générée automatiquement

To that goal we just simply filter restaurants following the date of their grades. The $elemMatch is used to select those who respect the following criteria : having a date whose date is later January the 1st 2013 and at least as early as January the 1st 2012 ($gte : greater than and equal to; $lt : less than).

Results :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

The visible restaurants are indeed one with a grade from 2012.

Une image contenant texte, Police, blanc, conception

Description générée automatiquement

## Query 6 : Count how many distinct streets containing at least one restaurant there are within a borough. (aggregation)

We should get the list of boroughs and the number of streets containing at least one restaurant. The rest of the streets doesn’t appear in our database, they cannot be taken into account.

[

{

$group: {

\_id: {

borough: "$borough",

street: "$address.street"

}

}

},

{

$group: {

\_id: "$\_id.borough",

count: { $sum: 1 }

}

},

{

$sort: { count: -1 }

}

]

In a first time we group by street and borough using $group and \_id. This step allows us to get rid of redundant streets.  
In a second time we sum the number of streets per borough.

Une image contenant texte, Police, capture d’écran, nombre

Description générée automatiquement

Results :

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

We obtain the expected list of boroughs as \_id and their respective count of restaurants.

## Query 7 : The restaurant who has the oldest grade (aggregation)

The expect result is a single restaurant as well as the date of its oldest grade.

[

{

$match: {

"grades.date": { $exists: true, $ne: null }

}

},

{

$addFields: {

oldestGrade: { $min: "$grades.date" }

}

},

{

$sort: {

oldestGrade: 1

}

},

{

$limit: 1

}

]

We first use $match to filter out the restaurants with no dated grade. We then add a field corresponding to the oldest dated grade a restaurant has with “addFields” for it to be usable. Then we sort them by ascending order of oldest grades for the expected result to be on top. Lastly we simply use $limit to only get the restaurant we are looking for.

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Result :

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

The oldest grade is from 2010 for the restaurant El Rancho Los Compadres!

## Query 8 : The street (and its borough) that holds the most restaurants. (aggregation)

We want to get the name of the street containing the greatest number of restaurants, as well as said number and its borough.

[

{

$group: {

\_id: { borough: "$borough", street: "$address.street" },

boroughName: { $first: "$borough" },

streetName: { $first: "$address.street" },

count\_of\_restaurants: { $sum: 1 }

}

},

{

$sort: { count\_of\_restaurants: -1 }

},

{

$limit: 1

}

]

The first thing we do id to group the restaurants by street and borough and then add the number of restaurants each of them has ($sum to get the number of element). We also create 2 new fields containing the naem Then we order them by descending order (-1) for the street with the most restaurants to be on top and lastly we single out the top using $limit.

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Result :

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

The result is, as expected, the names of the borough, the street and the number of restaurants in the street.

Broadway is the street with the greatest number of restaurants (615!) and is in Manhattan.

Complex Query

## Query : List of restaurants that had a good first review and a ‘bad’ last one

We expect to find all restaurants that were first nicely at first (an A grade) and a worse one at last.

Une image contenant texte, capture d’écran, Police, ligne

Description générée automatiquement

[

{

$unwind: "$grades"

},

{

$group: {

\_id: "$\_id",

name: { $first: "$name" },

initialGrade: { $first: "$grades.grade" },

initialDate: { $min: "$grades.date" },

latestGrade: { $last: "$grades.grade" },

latestDate: { $max: "$grades.date" }

}

},

{

$match: {

$expr: {

$and: [

{ $eq: ["$initialGrade", "A"] },

{ $in: ["$latestGrade", ["B", "C"]] }

]

}

}

}

]

The first step is to create a separate entry for each grade. We then group back restaurants by id and keep the first and last grades for each. Lastly, we select only the restaurants with an A as a first grade and a B or a C as a last one.

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

The result does contain the name of the restaurant, with the first and last grades as well as their date.

Quite a few restaurants seem to have lowered their quality over time.

Hard Query

## Query : The name and number of grades attributed to restaurants, order by the worst graded restaurants ever (aggregation) !

We want to find a list of all restaurants, and then order it in a way such that the one wit the most C grades comes first, in case of equality that with the most B ones, and so on for the A grades.

[{

$unwind: "$grades"

},

{

$group: {

\_id: "$\_id",

name: { $first: "$name" },

numA: { $sum: { $cond: [{ $eq: ["$grades.grade", "A"] }, 1, 0] } },

numB: { $sum: { $cond: [{ $eq: ["$grades.grade", "B"] }, 1, 0] } },

numC: { $sum: { $cond: [{ $eq: ["$grades.grade", "C"] }, 1, 0] } }

}

},

{

$sort: {

numC: -1,

numB: -1,

numA: -1

}

}

]  
  
The first step is to create a new entry for every grade (hence the $unwind), then to group back by restaurants($group and \_id: “$id”) and count their number of grades for each grades (number of As, of Bs and of Cs). The $eq is used to filter grades between them, $cond is used to apply that filter in order for a sum to be performed on the output.  
Lastly we sort the m by descending order following the sum of C grades first, then that of B ones and lastly the As.

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Results :

Here is a sample of 3 entries of our result:

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

As expected, the result is composed of the name and id of each restaurant, as well as their number of each grade. The first one has the greatest number of C grades (6>4) and the second one has more B than the third (3>1).

How is Chopstick still in business?