TP3 – NOSQL MONGODB RESTAURANT INSPECTIONS

Manon GARDIN

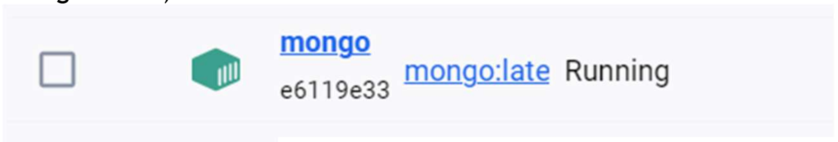
Matias OTTENSEN

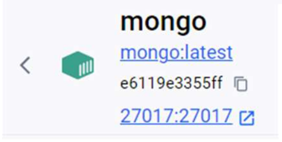
Alexandre GARNIER

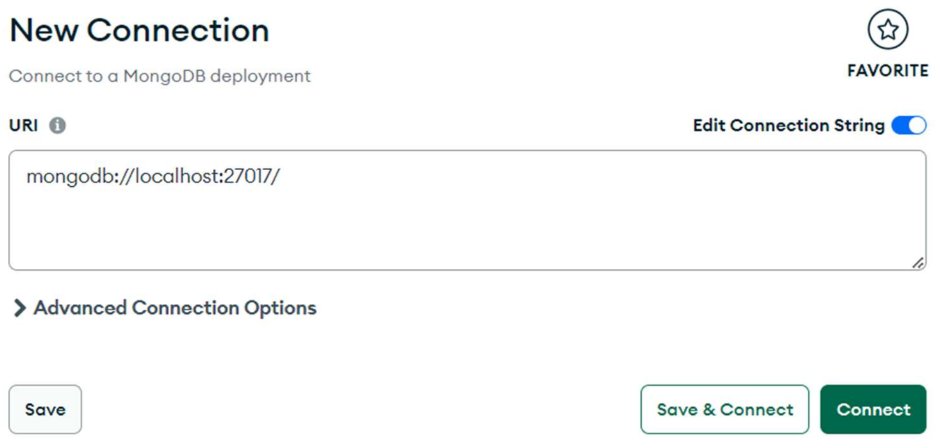
Tiphaine KACHKACHI

Create the Database

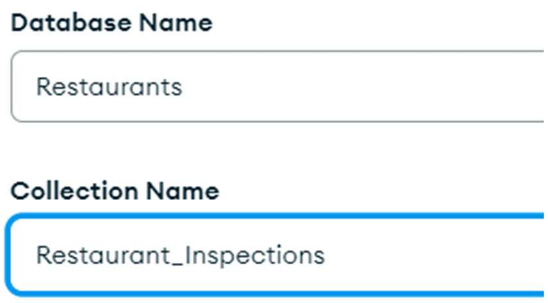
After running our container for MongoDb we can connect ourselves to mongodbCompass (or in the mongodb CLI)







```
mongodb://localhost:27017/
```

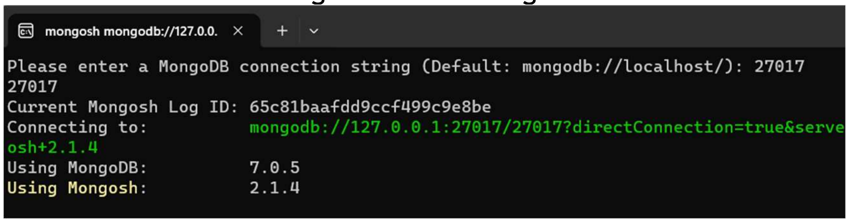

Database Name

Restaurants

Collection Name

Restaurant_Inspections

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

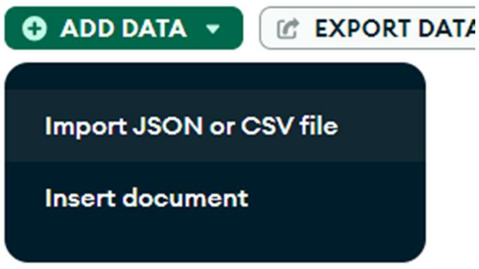


```
Please enter a MongoDB connection string (Default: mongodb://localhost/): 27017
27017
Current Mongosh Log ID: 65c81baafdd9ccf499c9e8be
Connecting to:          mongodb://127.0.0.1:27017/27017?directConnection=true&serve
osh+2.1.4
Using MongoDB:          7.0.5
Using Mongosh:          2.1.4
```

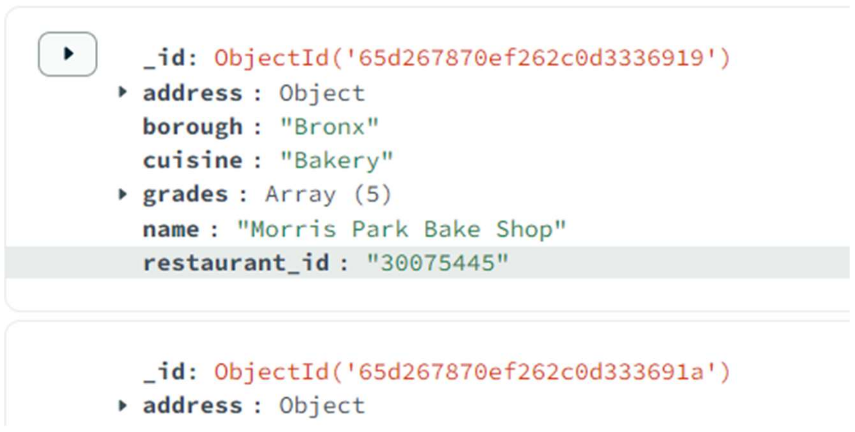
To connect ourselves to the database

Import the data

To import our data we can directly use the button :



All objects were added to our database !



```
_id: ObjectId('65d267870ef262c0d3336919')
address : Object
borough : "Bronx"
cuisine : "Bakery"
grades : Array (5)
name : "Morris Park Bake Shop"
restaurant_id : "30075445"

_id: ObjectId('65d267870ef262c0d333691a')
address : Object
```

Queries

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

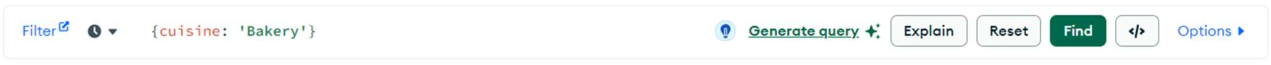
Easy Queries

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.

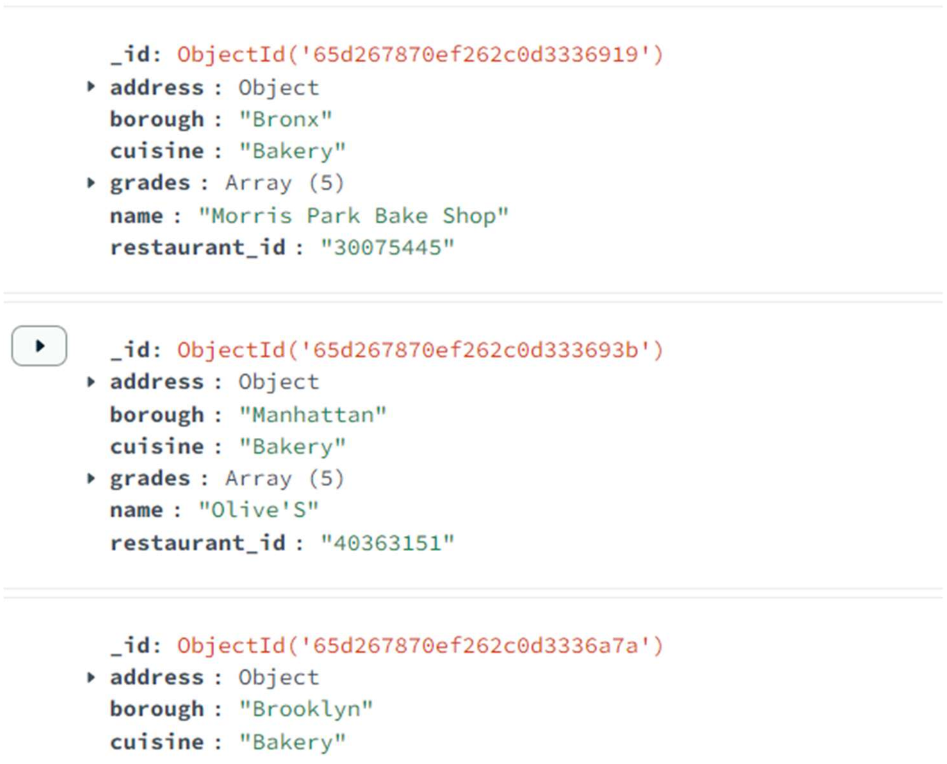
In the filter section :

{cuisine: 'Bakery'}



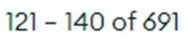
```
{cuisine: 'Bakery'}
```

Results :



```
  _id: ObjectId('65d267870ef262c0d3336919')
▸ address : Object
  borough : "Bronx"
  cuisine : "Bakery"
▸ grades : Array (5)
  name : "Morris Park Bake Shop"
  restaurant_id : "30075445"

  _id: ObjectId('65d267870ef262c0d333693b')
▸ address : Object
  borough : "Manhattan"
  cuisine : "Bakery"
▸ grades : Array (5)
  name : "Olive'S"
  restaurant_id : "40363151"

  _id: ObjectId('65d267870ef262c0d3336a7a')
▸ address : Object
  borough : "Brooklyn"
  cuisine : "Bakery"
```

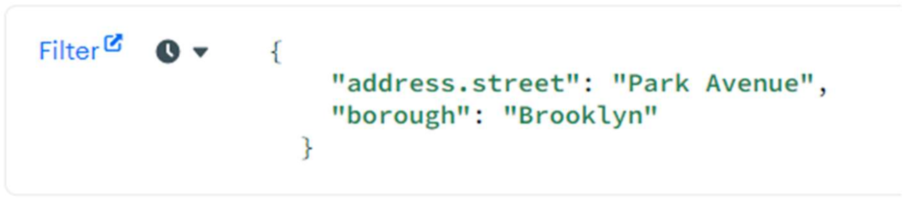
We got 691 results, as expected, we only find bakeries in the output.



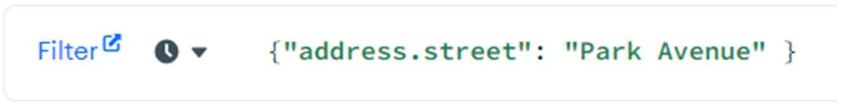
121 – 140 of 691

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.



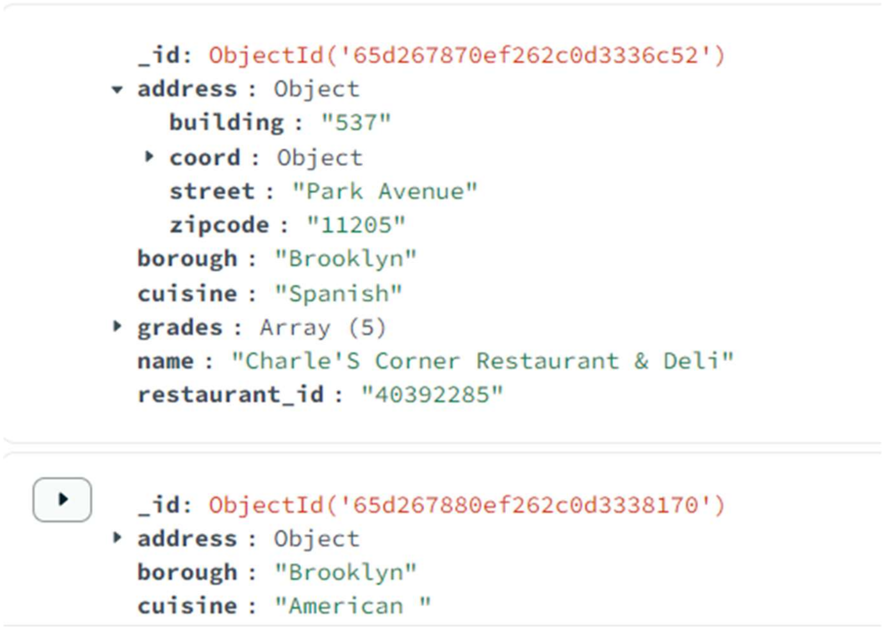
```
{
    "address.street": "Park Avenue",
    "borough": "Brooklyn"
  }
```



```
{"address.street": "Park Avenue" }
```

{
  "address.street": "Park Avenue",
  "borough": "Brooklyn"
 }

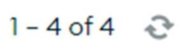
Results :



```
_id: ObjectId('65d267870ef262c0d3336c52')
address : Object
  building : "537"
  coord : Object
  street : "Park Avenue"
  zipcode : "11205"
borough : "Brooklyn"
cuisine : "Spanish"
grades : Array (5)
name : "Charle'S Corner Restaurant & Deli"
restaurant_id : "40392285"
```

```
_id: ObjectId('65d267880ef262c0d3338170')
address : Object
borough : "Brooklyn"
cuisine : "American "
```

We had few results, all of them are restaurants on Park Avenue in Brooklyn. Once again, we get all default fields as we simply filtered by location (borough and street).



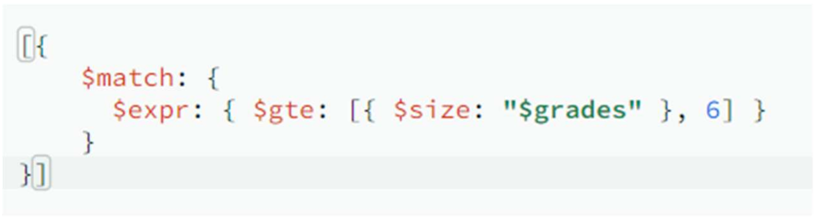
Query 3 : To find all restaurants who have been graded at least 6 times

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

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

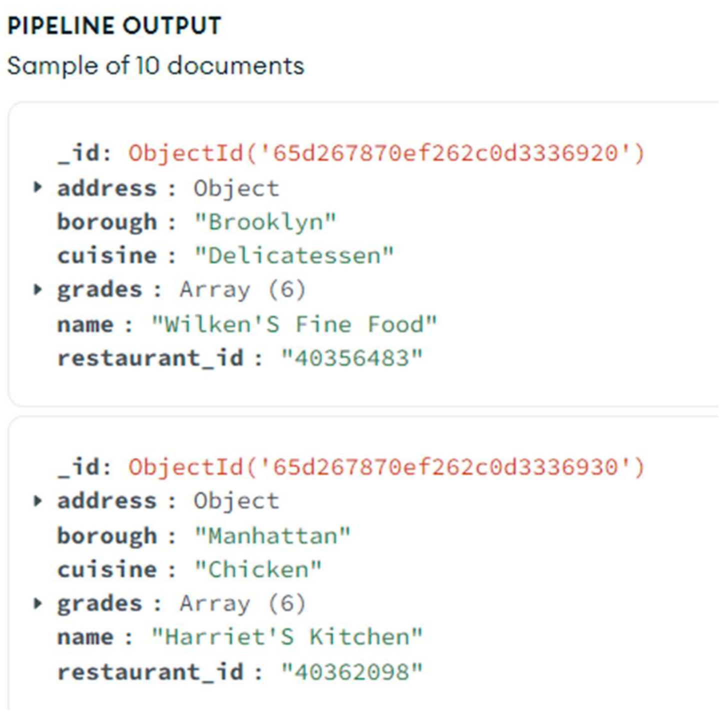
The $match is trying to find to find items who will be having the following characteristics. $size returns the number of elements in an array. Here we look for the numbers of grades in 'grades' and we want them to be equal of higher than 6. We do that with $gte which stands for 'greater than or equal' operator.

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



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

Results (sample displayed) :



PIPELINE OUTPUT
Sample of 10 documents

```
_id: ObjectId('65d267870ef262c0d3336920')
▸ address : Object
borough : "Brooklyn"
cuisine : "Delicatessen"
▸ grades : Array (6)
name : "Wilken'S Fine Food"
restaurant_id : "40356483"

_id: ObjectId('65d267870ef262c0d3336930')
▸ address : Object
borough : "Manhattan"
cuisine : "Chicken"
▸ grades : Array (6)
name : "Harriet'S Kitchen"
restaurant_id : "40362098"
```

We then observe than the result is composed of resaturants having been graded 6 times or more as expected, containing all others default fields as we didn't do anything special.

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

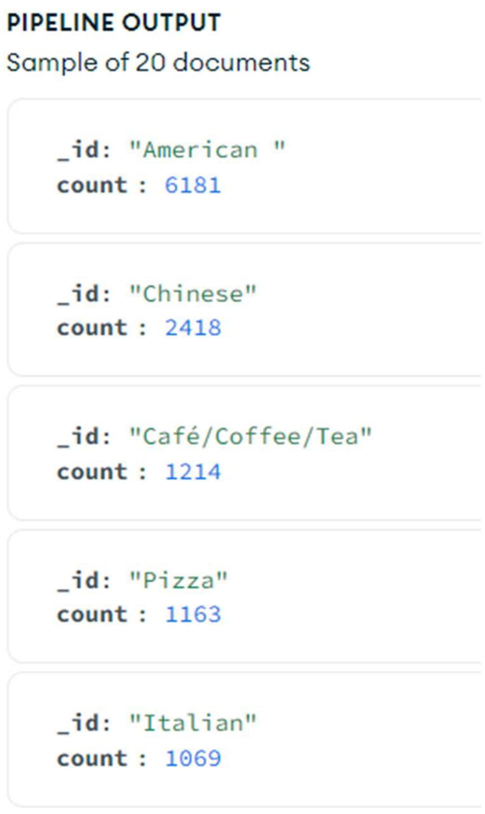
The expected output of this query is a list of all restaurants types as well as the number of restaurants available for each.

```
[
  {
    $group: {
      _id: "$cuisine",
      count: { $sum: 1 }
    }
  },
  {
    $sort: { count: -1 }
  }
]
```

```
[
  {
    $group: {
      _id: "$cuisine",
      count: { $sum: 1 }
    }
  },
  {
    $sort: { count: -1 }
  }
]
```

Firstly, $group to group the results by cuisine type (_id: "$cuisine”), its output should be a list of all cuisine types next to the sum of restaurants respecting such cuisine(count: { $sum: 1 }). 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) :


PIPELINE OUTPUT

Sample of 20 documents

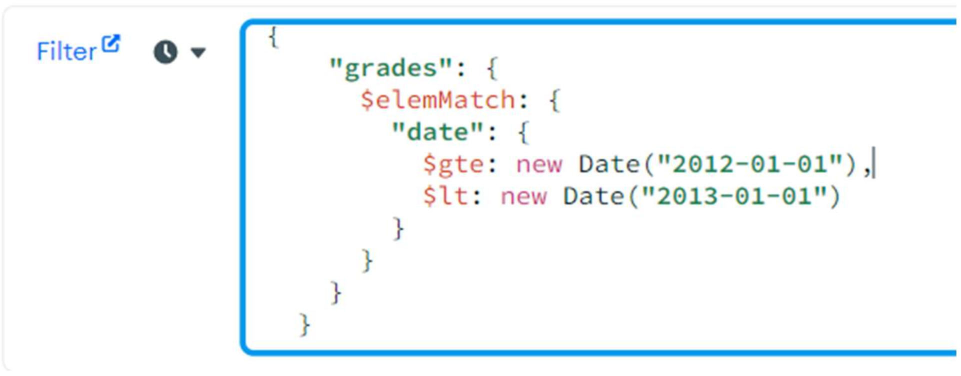
```
_id: "American "
count : 6181

_id: "Chinese"
count : 2418

_id: "Café/Coffee/Tea"
count : 1214

_id: "Pizza"
count : 1163

_id: "Italian"
count : 1069
```

The results are, as expected, composed of cuisine types in the _id field with the number of according restaurants in the count field.

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.

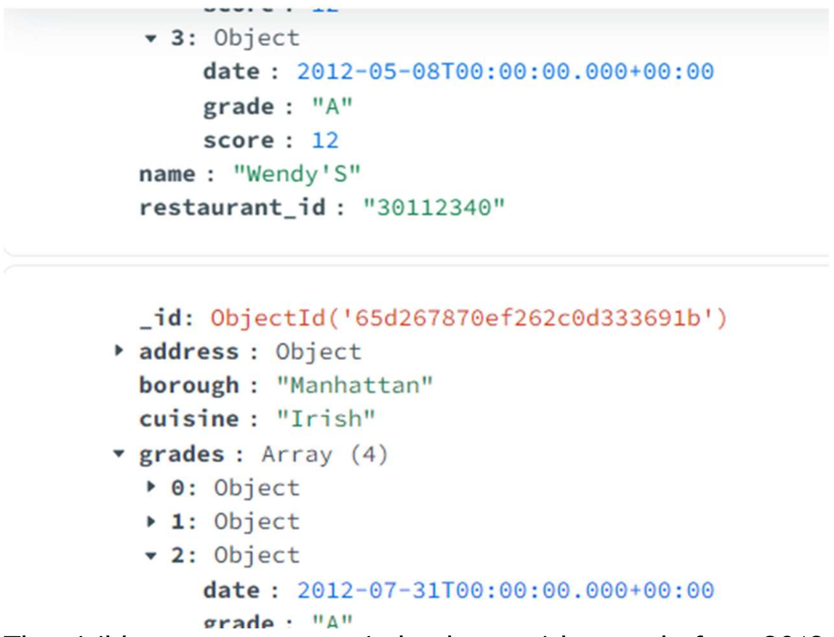
```
{
   "grades": {
     $elemMatch: {
       "date": {
         $gte: new Date("2012-01-01"),
         $lt: new Date("2013-01-01")
       }
     }
   }
 }
```



```
{
    "grades": {
      $elemMatch: {
        "date": {
          $gte: new Date("2012-01-01"),
          $lt: new Date("2013-01-01")
        }
      }
    }
  }
```

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 :



```
            score : 12
          ▾ 3: Object
              date : 2012-05-08T00:00:00.000+00:00
              grade : "A"
              score : 12
        name : "Wendy'S"
        restaurant_id : "30112340"

        _id: ObjectId('65d267870ef262c0d333691b')
      ▸ address : Object
        borough : "Manhattan"
        cuisine : "Irish"
      ▾ grades : Array (4)
          ▸ 0: Object
          ▸ 1: Object
          ▾ 2: Object
              date : 2012-07-31T00:00:00.000+00:00
              grade : "A"
```

The visible restaurants are indeed one with a grade from 2012. We also observe the creation of a new field date in the object category.

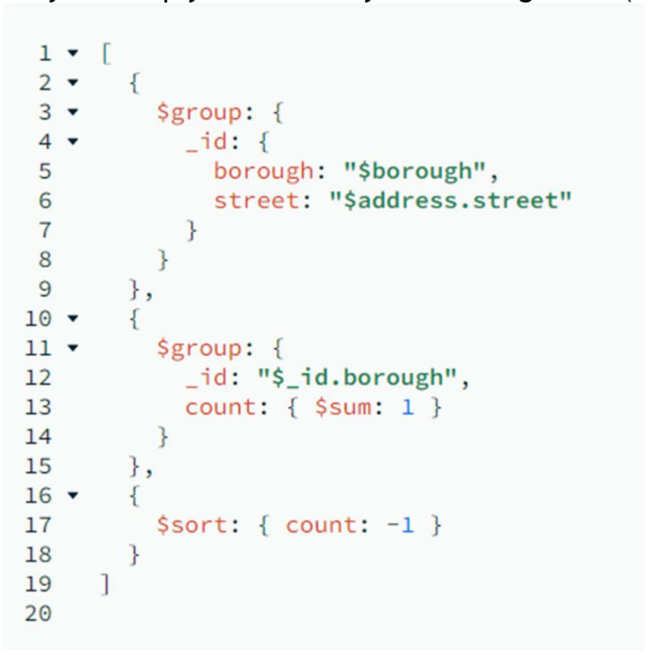
1 – 20 of 15825

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.

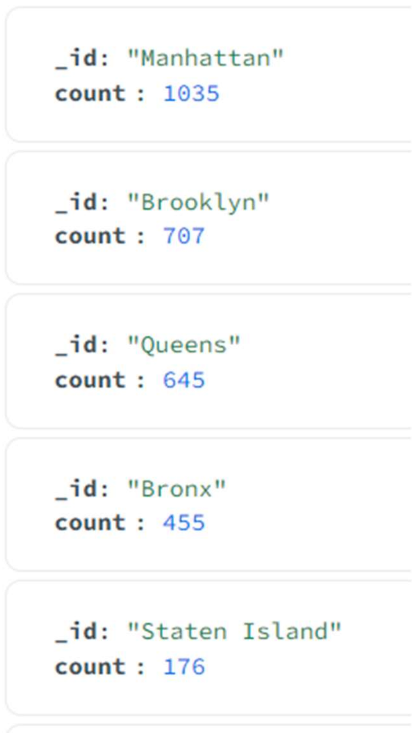
```
[
  {
    $group: {
      _id: {
        borough: "$borough",
        street: "$address.street"
      }
    }
  },
  {
    $group: {
      _id: "$_id.borough",
      count: { $sum: 1 }
    }
  },
  {
    $sort: { count: -1 }
  }
]
```

In a first time we group by both street and borough using $group and _id (id: { borough: "$borough", street: "$address.street" }). This step allows us to get rid of redundant occurrences of a single street. We also need to keep track of the borough to count the number of streets with a restaurant or mor they hold.
In a second time we group that result by borough to which we join their number of occurrences in the previous output, which corresponds to the number of corresponding streets they hold.
Lastly we simply sort them by descending order ($sort: { count: -1}).



```
[
  {
    $group: {
      _id: {
        borough: "$borough",
        street: "$address.street"
      }
    }
  },
  {
    $group: {
      _id: "$_id.borough",
      count: { $sum: 1 }
    }
  },
  {
    $sort: { count: -1 }
  }
]
```

Results :



```
_id: "Manhattan"
count : 1035

_id: "Brooklyn"
count : 707

_id: "Queens"
count : 645

_id: "Bronx"
count : 455

_id: "Staten Island"
count : 176
```

We obtain the expected list of boroughs as _id and their respective count of streets with at least one restaurant in the count field.

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

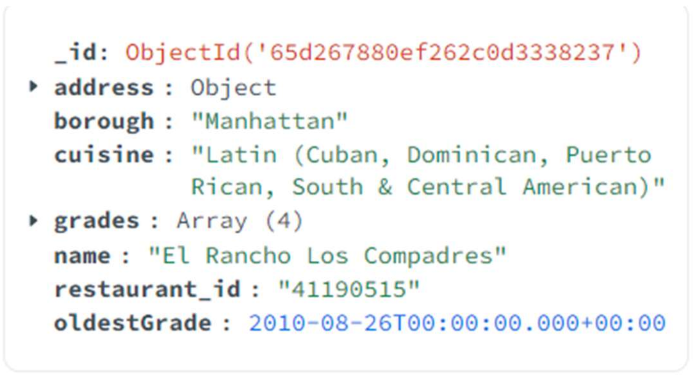
The expect result is a single restaurant as well as the date of its oldest grade.

```
[
  {
    $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.

```
[
  {
    $match: {
      "grades.date": { $exists: true, $ne: null }
    }
  },
  {
    $addFields: {
      oldestGrade: { $min: "$grades.date" }
    }
  },
  {
    $sort: {
      oldestGrade: 1
    }
  },
  {
    $limit: 1
  }
]
```

Result :



```
_id: ObjectId('65d267880ef262c0d3338237')
▸ address : Object
  borough : "Manhattan"
  cuisine : "Latin (Cuban, Dominican, Puerto
            Rican, South & Central American)"
▸ grades : Array (4)
  name : "El Rancho Los Compadres"
  restaurant_id : "41190515"
  oldestGrade : 2010-08-26T00:00:00.000+00:00
```

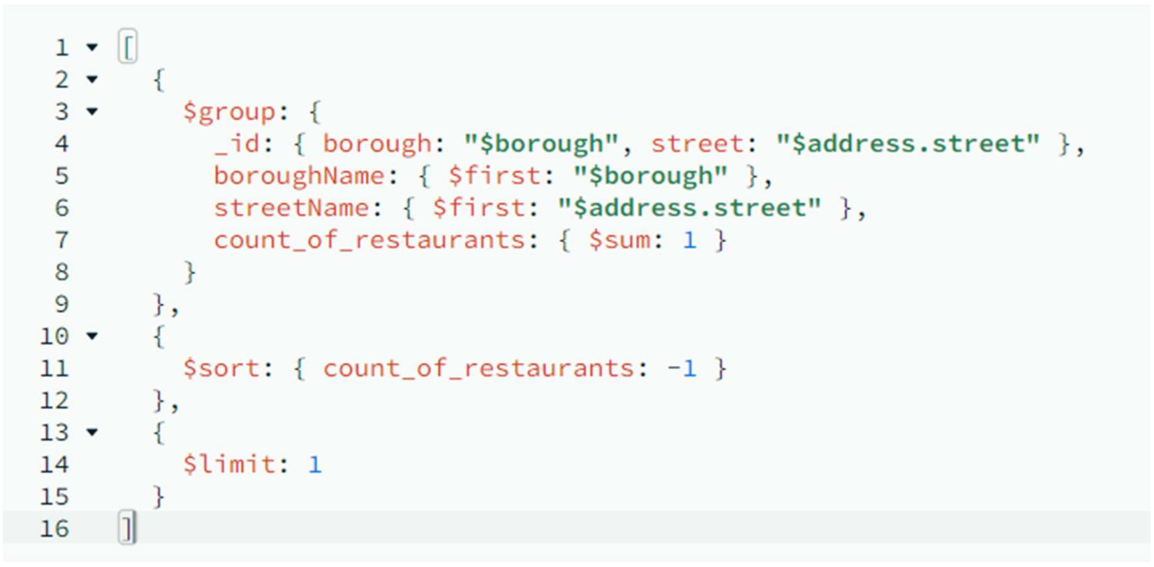
As expected, the result is a single restaurant with all related information as well as the extra oldestGrade field we added.

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.

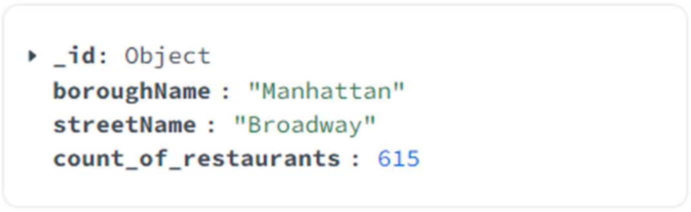
```
[
  {
    $group: {
      _id: { borough: "$borough", street: "$address.street" },
      boroughName: { $first: "$borough" },
      streetName: { $first: "$address.street" },
      count_of_restaurants: { $sum: 1 }
    }
  },
  {
    $sort: { count_of_restaurants: -1 }
  },
  {
    $limit: 1
  }
]
```

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 names of its borough and street. Then we order them by descending order of restaurant count (-1) for the street with the most restaurants to be on top and lastly we single out the top using $limit.



```
[
  {
    $group: {
      _id: { borough: "$borough", street: "$address.street" },
      boroughName: { $first: "$borough" },
      streetName: { $first: "$address.street" },
      count_of_restaurants: { $sum: 1 }
    }
  },
  {
    $sort: { count_of_restaurants: -1 }
  },
  {
    $limit: 1
  }
]
```

Result :



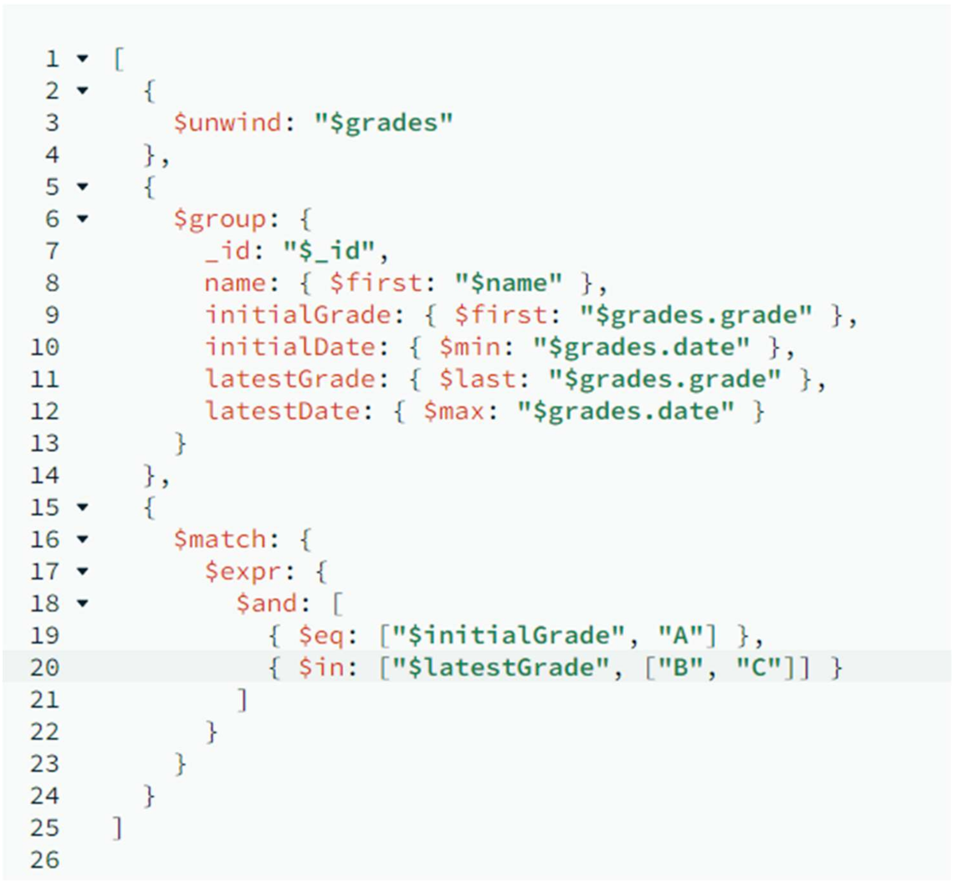
```
▸ _id: Object
  boroughName : "Manhattan"
  streetName : "Broadway"
  count_of_restaurants : 615
```

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

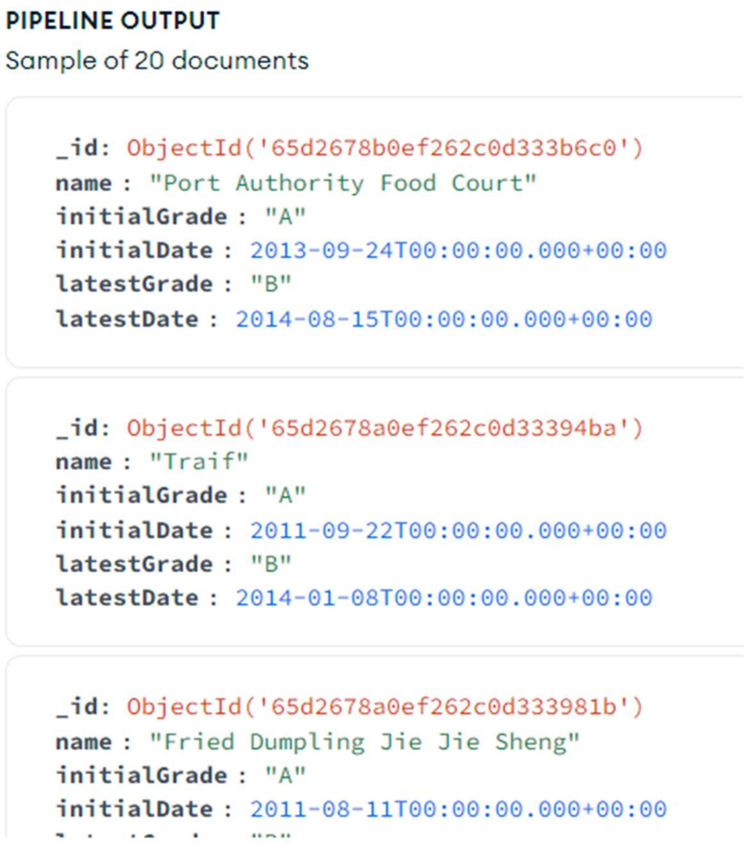
We expect to find all restaurants that were first nicely at first (an A grade) and a worse last one.



```
[
  {
    $unwind: "$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"]] }
        ]
      }
    }
  }
]
```

```
[
  {
    $unwind: "$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 as well as their name. Lastly, we select only the restaurants with an A as a first grade and a B or a C as a last one using a simple match. The output should then be a list of all restaurants matching this criteria containing their id in the _id field, as well as their name, their first grade with its date and their last one with its date.



PIPELINE OUTPUT
Sample of 20 documents

```
_id: ObjectId('65d2678b0ef262c0d333b6c0')
name : "Port Authority Food Court"
initialGrade : "A"
initialDate : 2013-09-24T00:00:00.000+00:00
latestGrade : "B"
latestDate : 2014-08-15T00:00:00.000+00:00

_id: ObjectId('65d2678a0ef262c0d33394ba')
name : "Traif"
initialGrade : "A"
initialDate : 2011-09-22T00:00:00.000+00:00
latestGrade : "B"
latestDate : 2014-01-08T00:00:00.000+00:00

_id: ObjectId('65d2678a0ef262c0d333981b')
name : "Fried Dumpling Jie Jie Sheng"
initialGrade : "A"
initialDate : 2011-08-11T00:00:00.000+00:00
```

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.

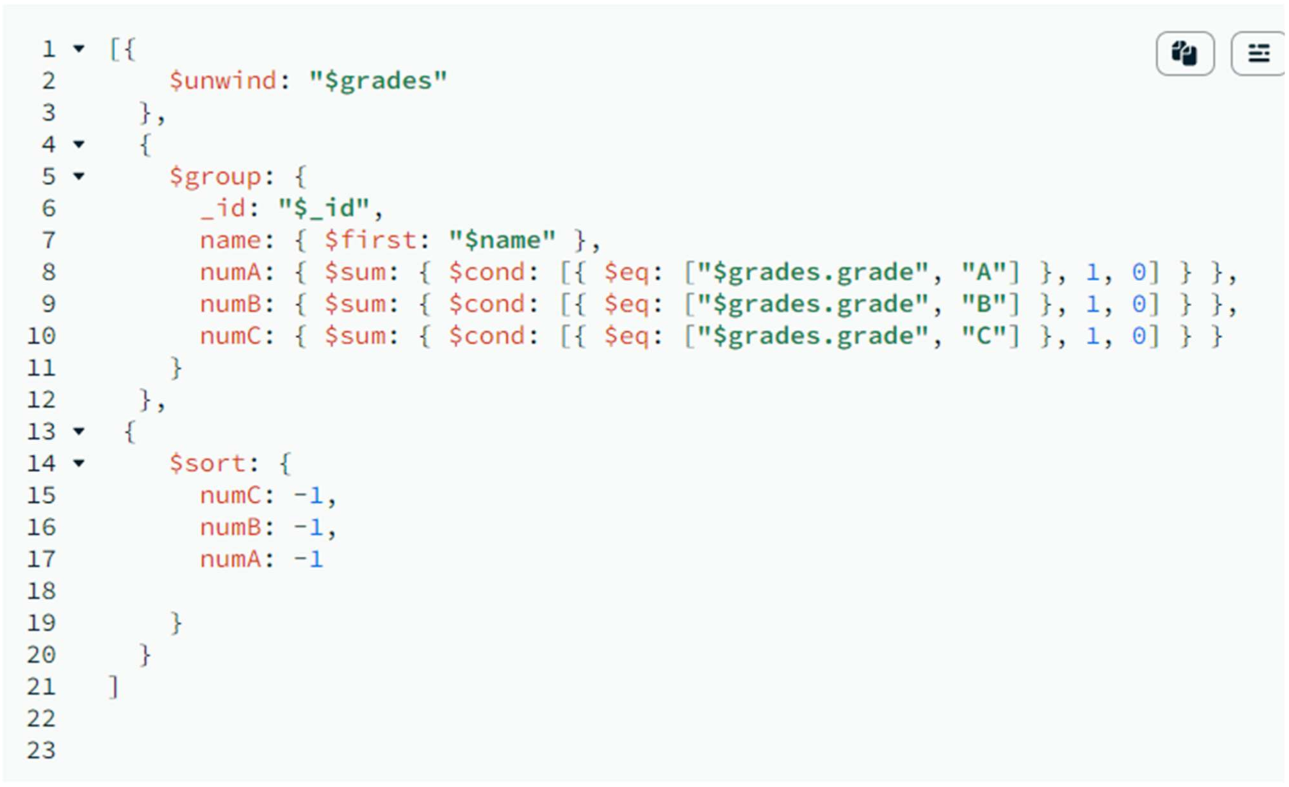
Hard Query

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 with 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

    }
  }
]
```

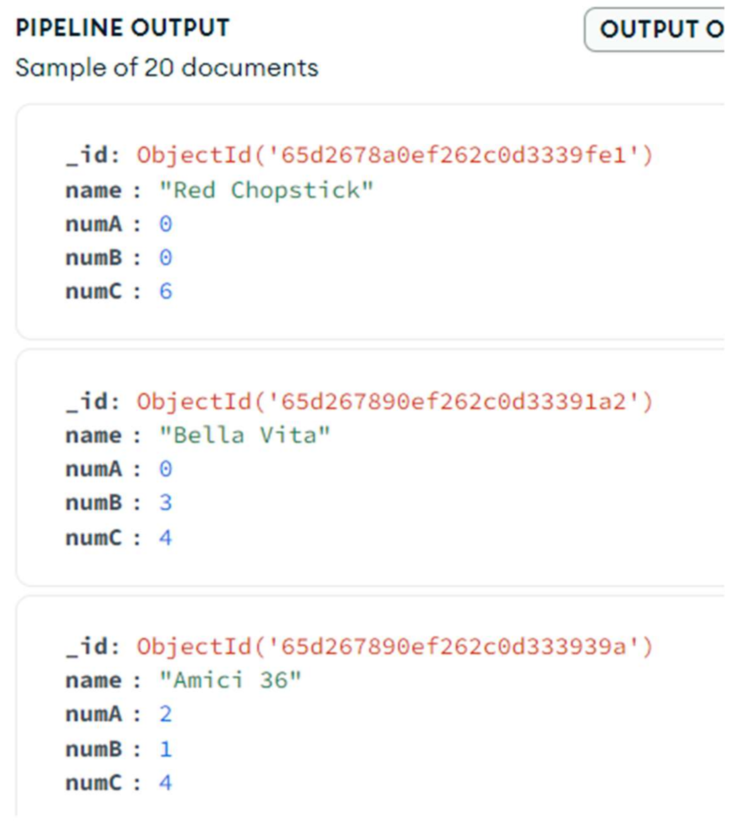
The first step is to create a new entry for every grade (hence the $unwind), then to group back by restaurant ($group and _id: “$id”) and count their number of grades for each grades (number of As, of Bs and of Cs). These are stored in the numA, numb and numC variables. A name variable is also created to keep track of their name.
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 them by descending order following the sum of C grades first, then that of B ones and lastly the As as planned.



```
[{
    $unwind: "$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

    }
  }
]
```

Results :

Here is a sample of 3 entries of our result:



Sample of 20 documents

```
_id: ObjectId('65d2678a0ef262c0d3339fe1')
name : "Red Chopstick"
numA : 0
numB : 0
numC : 6

_id: ObjectId('65d267890ef262c0d33391a2')
name : "Bella Vita"
numA : 0
numB : 3
numC : 4

_id: ObjectId('65d267890ef262c0d333939a')
name : "Amici 36"
numA : 2
numB : 1
numC : 4
```

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?