Emilio Gómez Berlanga

Certification Project

Donwload the database from https://www.mongodb.com/download-center/community
I chose the Enterprise server 3.4 version which is the more stable and common.
The installation process in windows has no complication as is a .msi file which installs the database by default in C:\Program Files\MongoDB\Server\3.4

Now we can access to the mongo shell, for that we open “Command“
and navigate to the **bin** folder on the installation path.

Then we are able to run scripts to set the database:
with **mongod** command a database will be created by default in the path C:\data\db and port 27017.
Adding the options **--path** and **--port** we can speciticate the path and port to our database.
Once the database is created and running we can access to it using the command **mongo**
Which by default will try to connect to http://127.0.0.1:27017. unless that we set other parameters.
From here we can access to the data by using the different mongo shell commands.

In the relations of One to Many we will chose the Document model. This data model uses embedded documents to create relationships between connected Data. Therefore we have that we will have a Collection **Category** which will contain **Brands** documents and this will contain **Items** documents.
In the other hand **Customer** could contain **Order** Documents, but as **Order** is also related to **Category** both collections will be referenced. However we will have an array of **Order** references inside Customer.
The relations Many to Many will be Modeled as references. This references will be between **Category** and **Customer** and **Category** and **Order**.

```
// Category Collection
[{
    "category_id": "CAT001",
    "category_name": "mobile",
    "brands": [{
                "brand_id": "BRN001",
                "brand_name": "Sony",
                "items":[{
                            "item id": "001",
                            "item name": "Xperia R1",
                            "price": 13990
                        }]
            }]
}]
```

```
// Customer Collection

[{
    "customer_id": "CUST001",
    "customer_name": "John",
    "contact_number": "9158385438",
    "address":
     {
         "streetAddress": "709 Honey Creek Dr.",

         "city": "New York",
         "state": "NY",
         "postalCode": "10028"
     },
    "birth_date": "12-12-1993",
    "orders": [ { "order_id": "ORD001" } ]
}]

//Order Collection
[{
    "order_id": "ORD001",
    "orderer_name": "John",
    "category": {
        "category_id": "CAT001",
        }
    "order_date": "12-12-2017",
    "shipping_date": "17-12-2017"
}]
```

Start the server:
start mongod --dbpath "C:\Users\Emilio\Documents\Edureka\04-MongoDB\Final Project\data" --logpath "C:\Users\Emilio\Documents\Edureka\04-MongoDB\Final Project\logs\db.log" --port 40000 --logappend

Create the database:

use Mongo_DB_Project

Create the user:
db.createUser({

user: "
Adam
",
pwd: "adam
",
roles: ["readWrite","dbAdmin"]
})

To start the server with authentication we start it like in the first step but with --auth option

Then connect to the server with authentication:

mongo --port 40000 -u Adam -p adam --authenticationDatabase Mongo_DB_Project

Insert the data into the database

>mongoimport --db Mongo_DB_Project --collection Orders --file "C:\Users\Emilio\Documents\Edureka\04-MongoDB\Final Project\P3 Orders.json" --jsonArray --port 40000 -u Adam -p adam

>mongoimport --db Mongo_DB_Project --collection Customers --file "C:\Users\Emilio\Documents\Edureka\04-MongoDB\Final Project\P3 Customers.json" --jsonArray --port 40000 -u Adam -p adam

>mongoimport --db Mongo_DB_Project --collection Categories --file "C:\Users\Emilio\Documents\Edureka\04-MongoDB\Final Project\P3 Categories.txt" --jsonArray --port 40000 -u Adam -p adam

To Update the price of the Item “Dell OptiPlex” first we find the Document that contains it

db.Categories.find({'brands.items.item name':"Dell OptiPlex"})

```
{ "_id" : ObjectId("5e77ab0d04ff621e50b29909"),
    "category_id" : "CAT002",
    "category_name" : "computer",
    "brands" : [ { "brand_id" : "BRN003",
                   "brand_name" : "Dell",
                   "items" : [ { "item id" : "009",
                                 "item name" : "Dell OptiPlex",
                                 "price" : 32200
                               },
                               { "item id" : "010",
                                 "item name" : "Dell Inspiron",
                                 "price" : 26990
```

```
                                },
                                { "item id" : "011",
                                    "item name" : "Dell AIO",
                                    "price" : 23876
                                } ]
                    },
                    { "brand_id" : "BRN004",
                       "brand_name" : "Apple",
                       "items" : [ { "item id" : "013",
                                        "item name" : "Apple iMac",
                                         "price" : 82000
                                },
                                { "item id" : "014",
                                    "item name" : "Apple MGEN2HN/A",
                                    "price" : 52849
                                } ]
                   } ]
}
```

using this filter we can do the update:

db.Categories.update({"brands.items.item name":"Dell OptiPlex"},{$set:{"brands.0.items.$.price":33000}},false,true)

To find duplicated Orders with different order_id will use the aggregate command:

db.Orders.aggregate({"$group":{"_id":"$orderer_name","count":{"$sum":1}}})

The result shows us how much orders we have for the same orderer_name:

{ "_id" : "Smith", "count" : 1 }
{ "_id" : "James", "count" : 1 }
{ "_id" : "John", "count" : 2 }

We noticed that “John” has two orders, but that doesn’t mean they have to be duplicated.

MongoDB Enterprise > db.Orders.find({"orderer_name":"John"})

{ "_id" : ObjectId("5e77aa7d04ff621e50b298f0"), "order_id" : "ORD001", "orderer_name" : "John", "category" : { "category_id" : "CAT001", "category_name" : "mobile" }, "brand" : { "brand_id" : "BRN002", "brand_name" : "Samsung" }, "items" : { "item id" : "005", "item name" : "Samsung Galaxy J7", "price" : 11890 }, "order_date" : "12-12-2017", "shipping_date" : "17-12-2017" }

{ "_id" : ObjectId("5e77aa7d04ff621e50b298f2"), "order_id" : "ORD003", "orderer_name" : "John", "category" : { "category_id" : "CAT001", "category_name" : "mobile" }, "brand" : { "brand_id" : "BRN002", "brand_name" : "Samsung" }, "items" : { "item id" : "005", "item name" : "Samsung Galaxy J7", "price" : 11890 }, "order_date" : "12-12-2017", "shipping_date" : "17-12-2017" }

After getting the two orders and compare them we can see that they are both the same, only the Order_id is different.

As we are noticed that is a duplicated order we proceed to delete it:

```
db.Orders.deleteOne( { order_id: "ORD003" } )
```

Now we are going to do a Backup of the database. For that we exit from the mongo shell to return to the command-line where we will use the mongoexport tool to create a JSON file of the data. We have to create one file for each Collection:

```
mongoexport --port 40000 --db Mongo_DB_Project --out
"C:\Users\Emilio\Documents\Edureka\04-MongoDB\Final
Project\backup\Categories_backup.json" --collection Categories -u Adam -p adam

mongoexport --port 40000 --db Mongo_DB_Project --out
"C:\Users\Emilio\Documents\Edureka\04-MongoDB\Final
Project\backup\Orders_backup.json" --collection Orders -u Adam -p adam

mongoexport --port 40000 --db Mongo_DB_Project --out
"C:\Users\Emilio\Documents\Edureka\04-MongoDB\Final
Project\backup\Customers_backup.json" --collection Customers -u Adam -p adam
```

To import again the data we only have to import the files like this, repeating the process for each collection-file:

```
mongoimport --db Mongo_DB_Project --collection Categories --file
"C:\Users\Emilio\Documents\Edureka\04-MongoDB\Final
Project\backup\Customers_backup.json" --jsonArray --port 40000 -u Adam -p adam
```

To improve performance we are going to create some Indexes. The more logical first step would be create Single field indexes for the id of each collection which will help when looking for specific documents:

```
db.Categories.createIndex({category_id:1})
db.Customers.createIndex({customer_id:1})
db.Orders.createIndex({order_id:1})
```

After that it could be also interesting to create a Multi Index for the brands inside Categories to find faster results from this array.

Emilio Gómez Berlanga

db.Categories.createIndex({category_id:1,”brands.brand_id”:1})

To know how much price

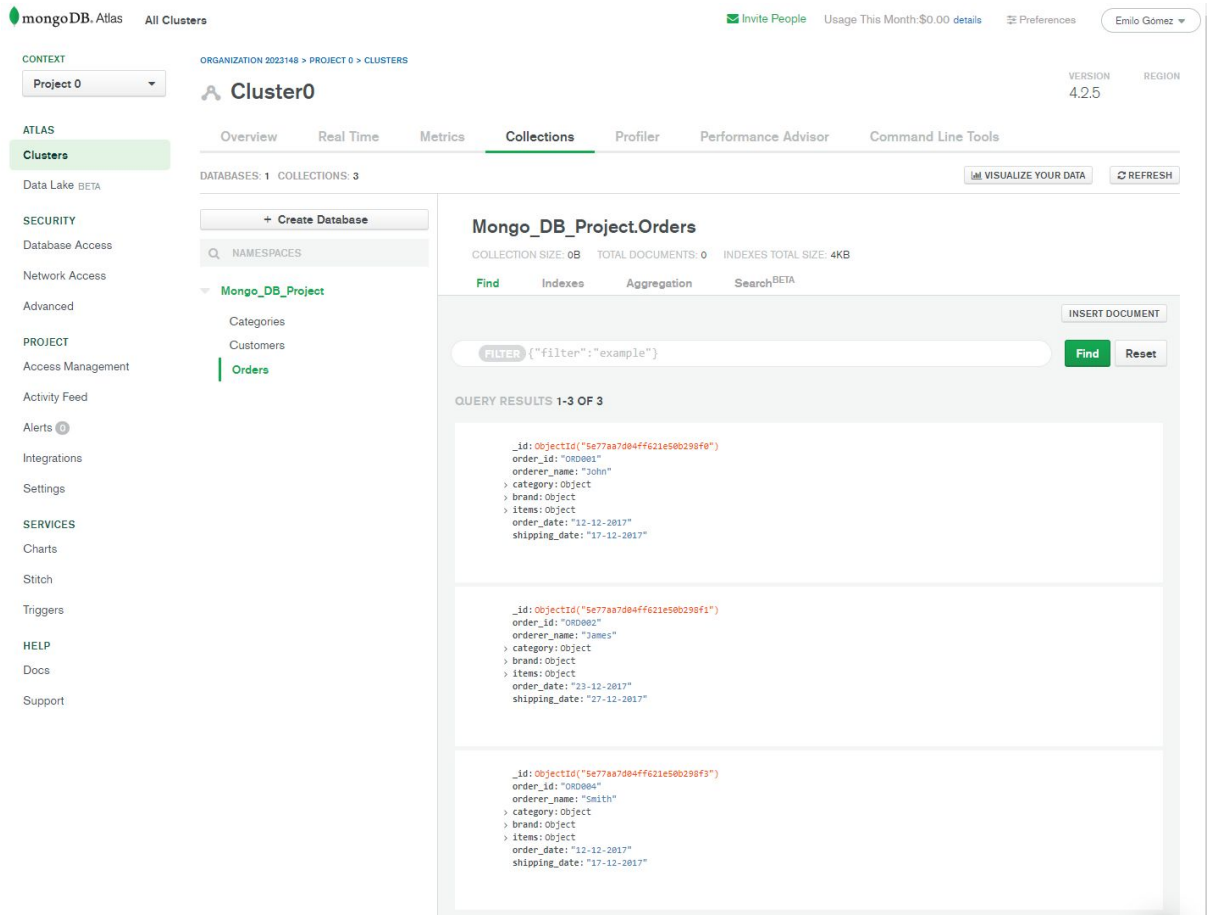
db.Orders.aggregate({"$group":{_id:{},total:{$sum:"$items.price"}}})
{ "_id" : { }, "total" : 109889 }

To create a cluster in MongoDB Atlas we go to the web page and register. Choose the service we want to contract. In our case Cluster0.

once we have created a cluster we can connect to it through the Shell like this:

mongo "mongodb+srv://cluster0-xj6hy.mongodb.net/test"

We are going to create an Stich application where to store the data.
In the Snitch application we create the Database and Collections.
We can import the backup files through the shell or using the web management of the Atlas Database.



Emilio Gómez Berlanga

MongoDB has built-in diagnosis tools:

- **mongostat:** which provides a quick overview of the server instance, quickly spot-check database activity, returns cache statistics and other sort of views.
- **mongotop:** is a tool that provides a method to track amount of time of reading/writing data.
- **mongoreplay:** Consists in a traffic capture and replay tool used to inspect and record commands that can be accessed by other host. Helps to preview perform under different environments, investigate an issue or database activity.
- **mongoperf:** Used to check disk performance, schedule tests of disk.

To check about invalid collections we could use the following commands:

db.runCommand({dbstats: 1, scale:1024})
db.runCommand({collstats: "Orders", scale:1024})

the first returns storage statistics of the database and the second of a specific collection.

and here an example of the mongostat tool:

c:\Program Files\MongoDB\Server\3.4\bin>mongostat --port 40000 -u Adam -p adam --authenticationDatabase Mongo_DB_Project
insert query update delete getmore command dirty used flushes vsize res qrw arw net_in net_out conn time
*0 *0 *0 *0 0 497|0 0.0% 0.0% 0 4.95G 120M 0|0 1|0 39.3k 11.8m 1 Apr 19 19:07:16.340
*0 *0 *0 *0 0 2|0 0.0% 0.0% 0 4.95G 120M 0|0 1|0 162b 48.7k 1 Apr 19 19:07:17.315
*0 *0 *0 *0 0 1|0 0.0% 0.0% 0 4.95G 120M 0|0 1|0 157b 47.5k 1 Apr 19 19:07:18.317
*0 *0 *0 *0 0 2|0 0.0% 0.0% 0 4.95G 120M 0|0 1|0 158b 47.6k 1 Apr 19 19:07:19.316
*0 *0 *0 *0 0 2|0 0.0% 0.0% 0 4.95G 120M 0|0 1|0 158b 47.5k 1 Apr 19 19:07:20.315
*0 *0 *0 *0 0 1|0 0.0% 0.0% 0 4.95G 120M 0|0 1|0 157b 47.5k 1 Apr 19 19:07:21.316
*0 *0 *0 *0 0 2|0 0.0% 0.0% 0 4.95G 120M 0|0 1|0 158b 47.6k 1 Apr 19 19:07:22.316
*0 *0 *0 *0 0 1|0 0.0% 0.0% 0 4.95G 120M 0|0 1|0 157b 47.5k 1 Apr 19 19:07:23.316
*0 *0 *0 *0 0 1|0 0.0% 0.0% 0 4.95G 120M 0|0 1|0 157b 47.5k 1 Apr 19 19:07:24.316
*0 *0 *0 *0 0 1|0 0.0% 0.0% 0 4.95G 120M 0|0 1|0 157b 47.5k 1 Apr 19 19:07:25.316

For warnings and errors while performing tasks would be useful mongoreplay to reproduce and investigate the issues the users could have.

Emilio Gómez Berlanga

To create the replica set first we start a database as usual but with the --replSet parameter and value “rs” which will be the name of our replica set.

start mongod --replSet rs --dbpath "C:\Users\Emilio\Documents\Edureka\04-MongoDB\Final Project\datars\rs1" --logpath "C:\Users\Emilio\Documents\Edureka\04-MongoDB\Final Project\logs\db-rs1.log" --port 40001 --logappend

followed we create the other two servers changing only the data path, the name of the log file and the port.

start mongod --replSet rs --dbpath "C:\Users\Emilio\Documents\Edureka\04-MongoDB\Final Project\datars\rs2" --logpath "C:\Users\Emilio\Documents\Edureka\04-MongoDB\Final Project\logs\db-rs2.log" --port 40002 --logappend

start mongod --replSet rs --dbpath "C:\Users\Emilio\Documents\Edureka\04-MongoDB\Final Project\datars\rs3" --logpath "C:\Users\Emilio\Documents\Edureka\04-MongoDB\Final Project\logs\db-rs3.log" --port 40003 --logappend

Then we connect as usual to the first database:

mongo --port 40001

And execute the command rs.initiate() which will set the actual server as the primary server of the replica set.

Next we add the other two servers to the replica set with the following command:

rs.add(“LOCALHOST:40002”)
rs.add(“LOCALHOST:40003”)

The replica set is ready.