INSTITUTO POLITÉCNICO DE BRAGANÇA
Escola Superior de Tecnologia e de Gestão

Bases de Dados | Databases
2025/26

Caderno de Exercícios | Exercise Book

MongoDB

@jprp

2025

INSTITUTO POLITÉCNICO DE BRAGANÇA
Escola Superior de Tecnologia e de Gestão

Índice

Exercício 1 | Install MongoDB Compass - localhost

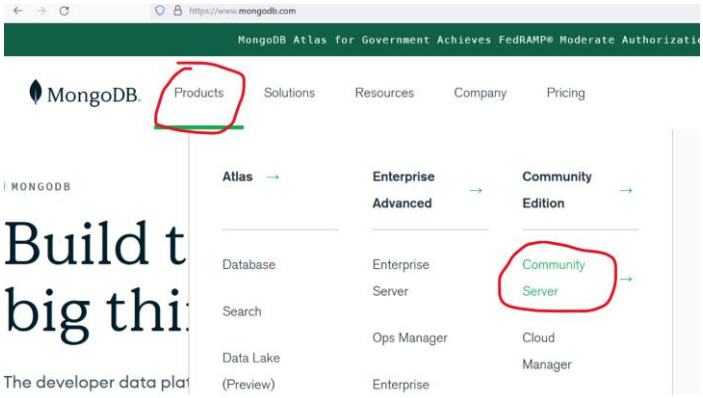
Open up MongoDB Compass to begin. If you click Connect without entering any information, Compass will automatically attempt to connect to a local MongoDB server running with the default configuration. Click Connect to connect to the MongoDB serv

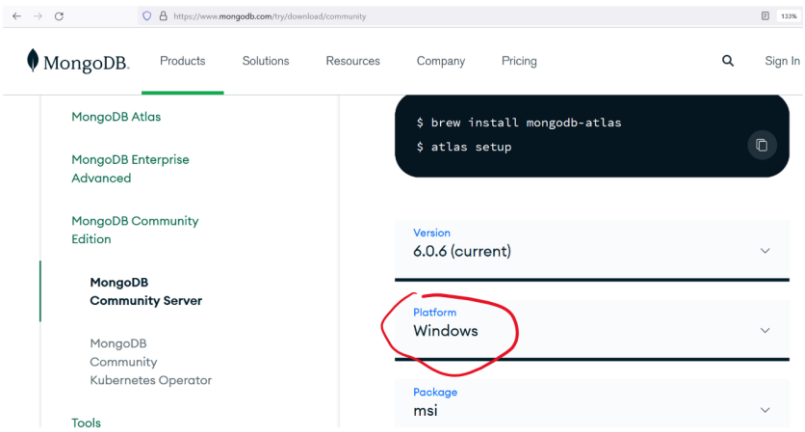
In MongoDB Compass, you create a database and add its first collection at the same time:

- Click "Create Database" to open the dialog.
- Enter the name of the database and its first collection.
- Click "Create Database"

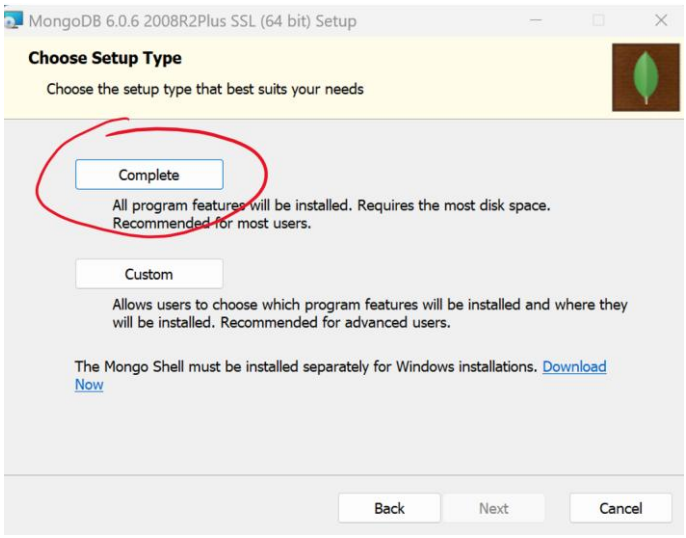
To start with the MongoDB Compass, **you need to download the MongoDB Server and install it**

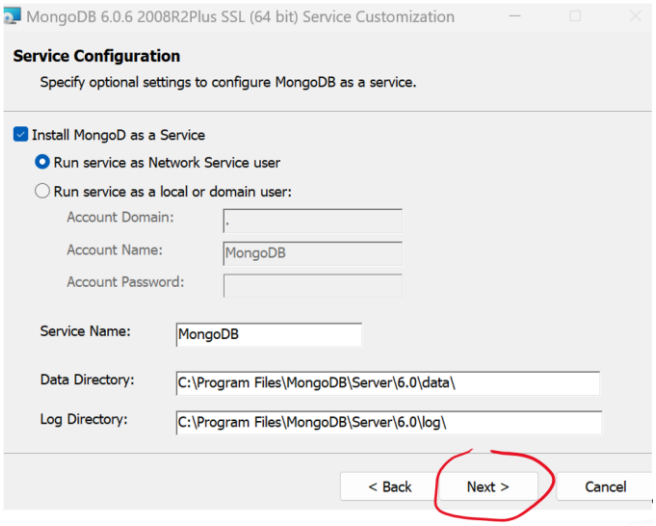
1) Download “Community Server”



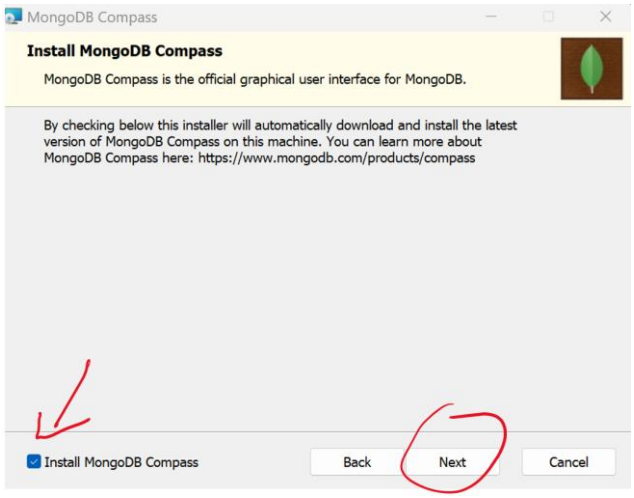


2) Installing MongoDB Community Server and MongoDB Compass



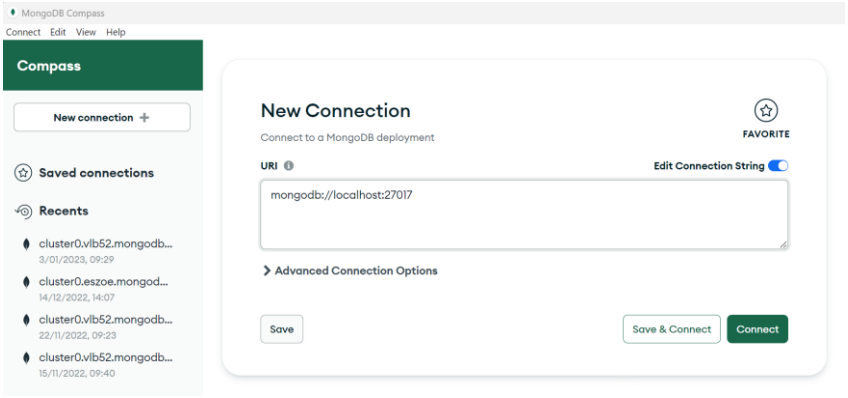


a) Select option to install MongoDB Compass.



3) Connect to the server

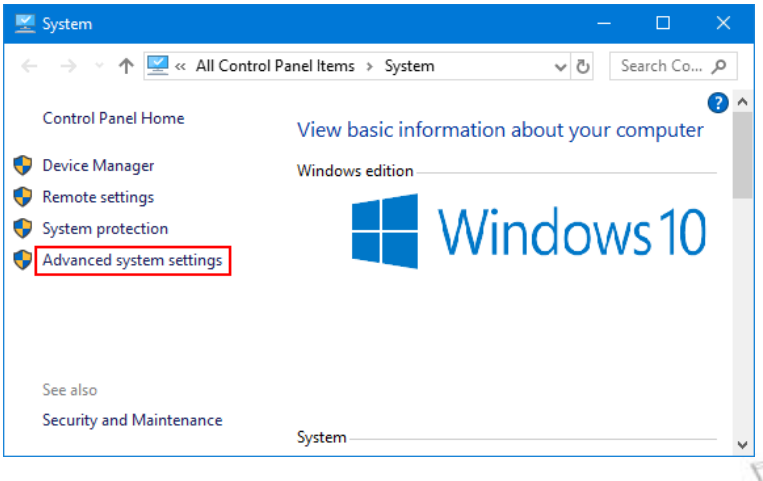
mongodb://localhost:27017



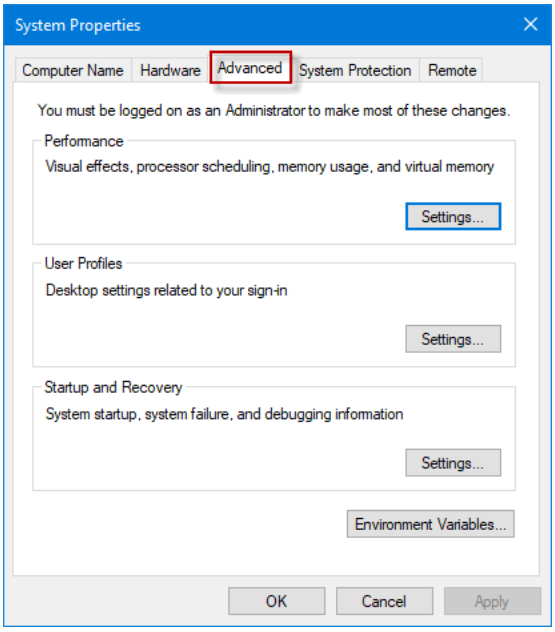
4) Environment Variables (variáveis de ambiente)

a) After successful installation, Right-click on 'This PC' or 'My Computer'. Choose properties

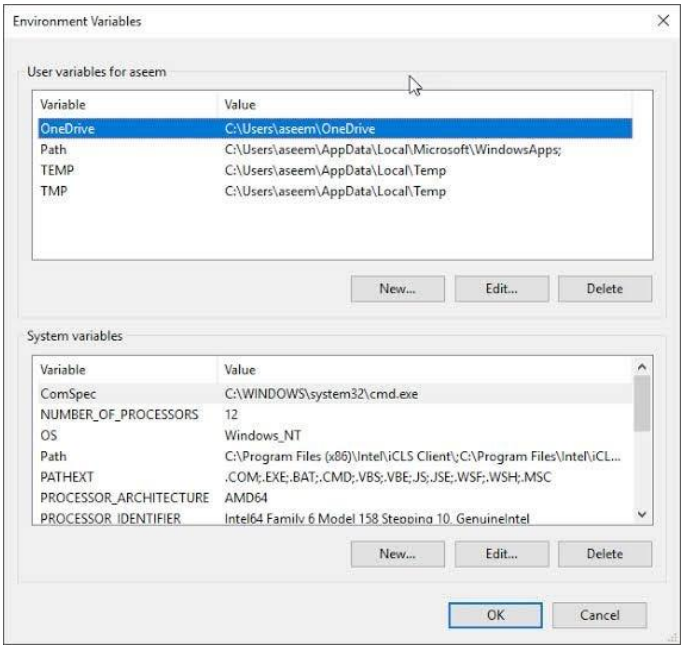
b) Choose the `advance system setting` options



c) Click on Environment Variables under Advance section.

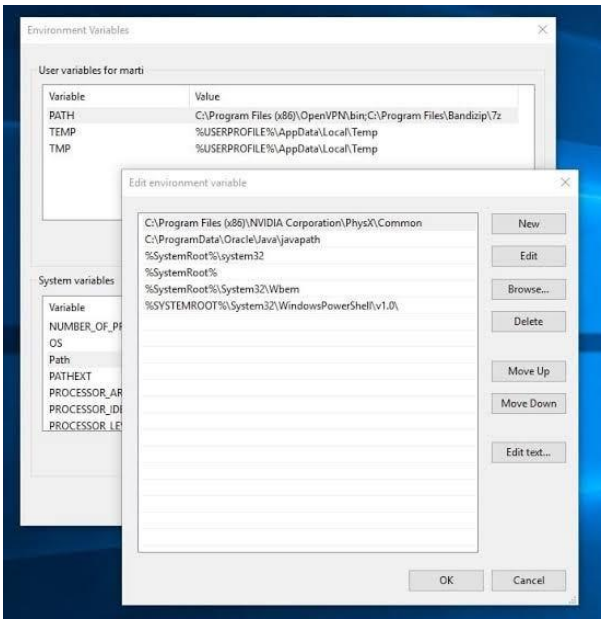


d) Choose Path value under system variables and click Edit button



e) Now get your mongo path to your system, where your MongoDB is installed. For example, if you installed MongoDB in C drive, then it your path will be like this: `C:\Program Files\MongoDB\Server\VERSION\bin`

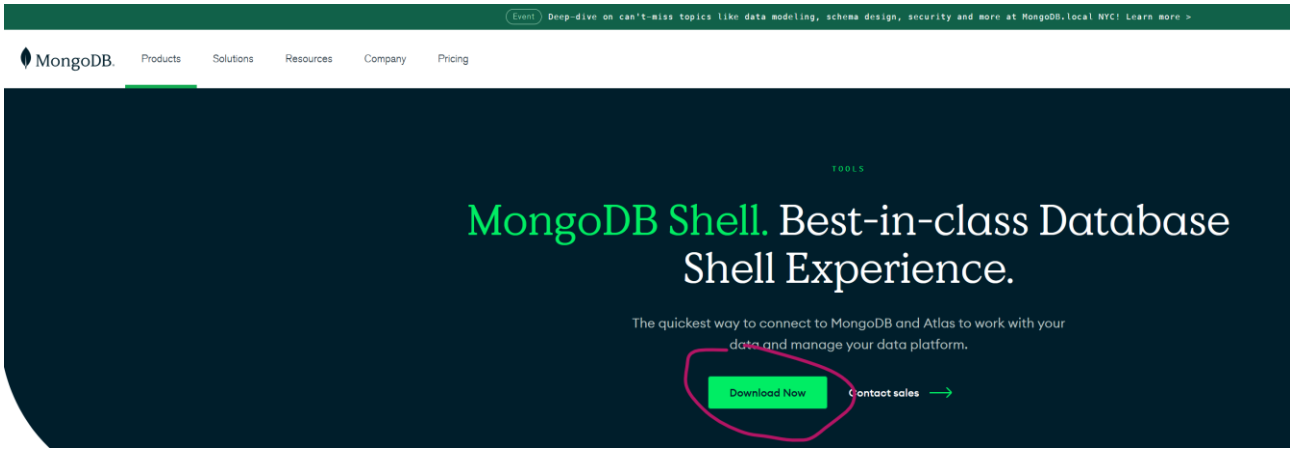
f) Copy this path and enter as a new environment value on Edit environment variables page



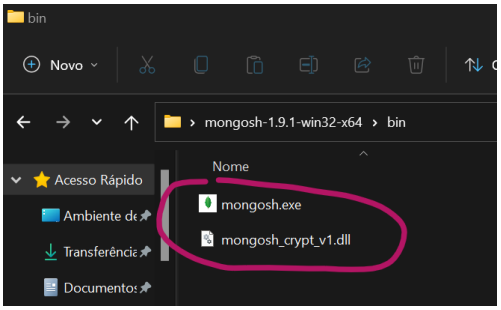
g) Now click on OK and close all active dialog box. Your environment is set, restart your terminal and now enter mongo , it will open mongo-shell

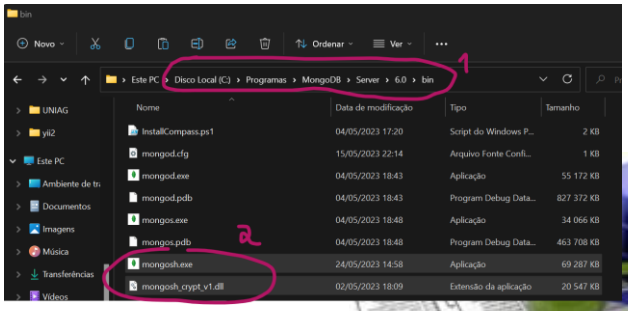
5) Install MongoDB Shell (command line)

As of version 6 of MongoDB, the command line (Shell) has to be installed separately and is called Mongo Shell: https://www.mongodb.com/products/shell



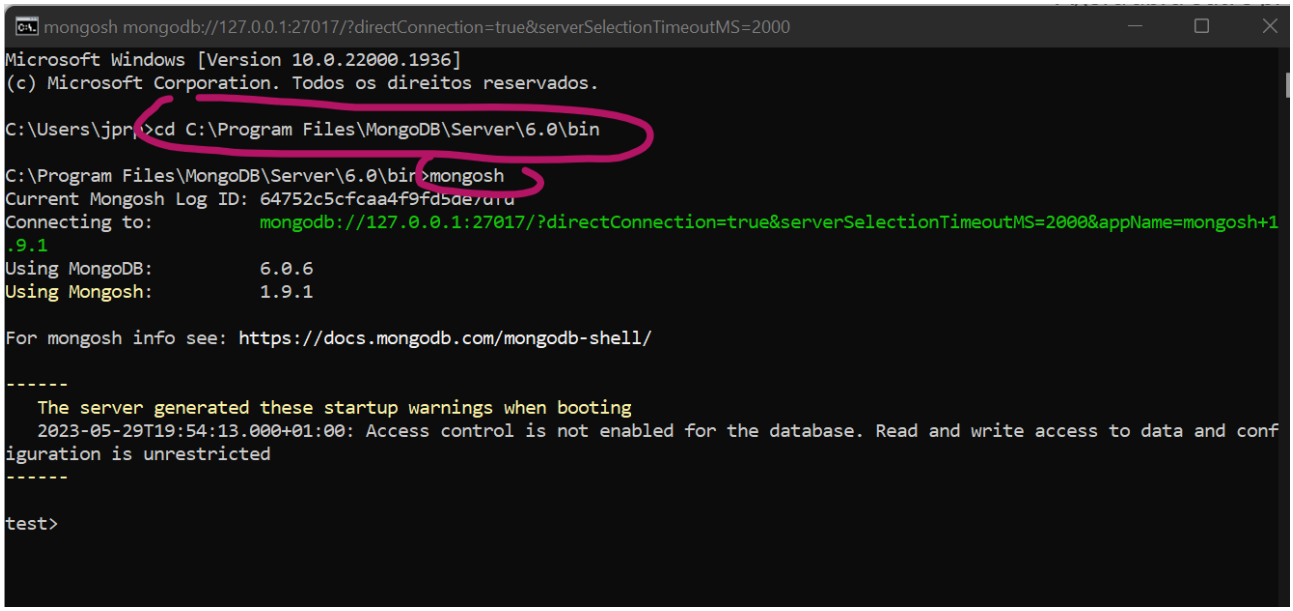
After downloading (zip), they should copy the contents of the bin folder to C:\Program Files\MongoDB\Server\6.0\bin





Open the command line (cmd) and go to the mongoDB bin folder

Inside the Mongo bin folder, type "mongosh" to connect to the local server (mongodb://127.0.0.1:27017):

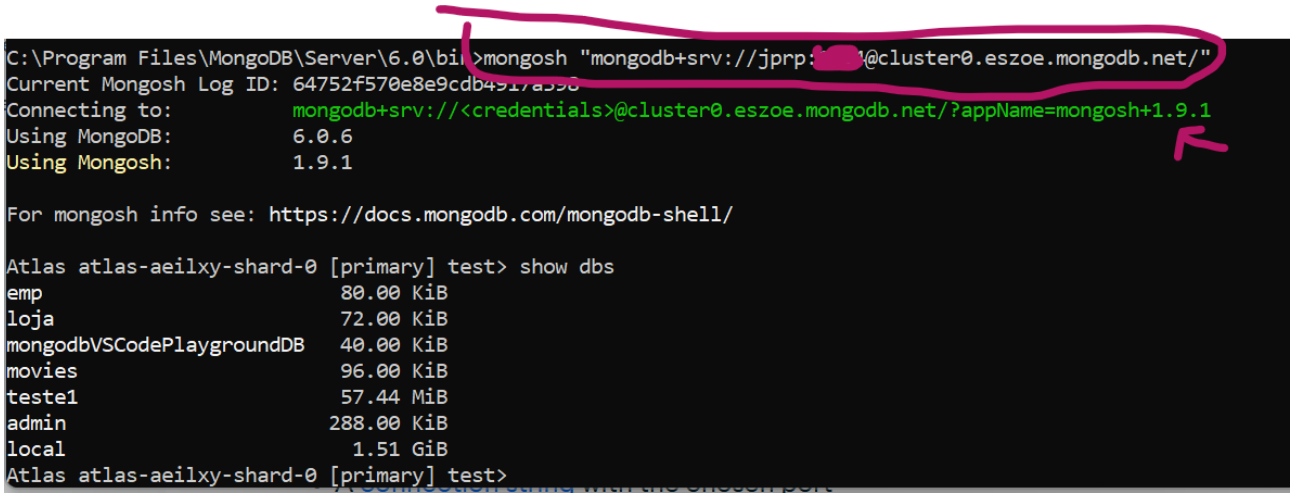


The command below in the client terminal shows the databases on that server:

> show databases

a) Connect to mongoDB atlas cloud

mongosh "mongodb+srv://jprp:<password>@cluster0.eszoe.mongodb.net/"



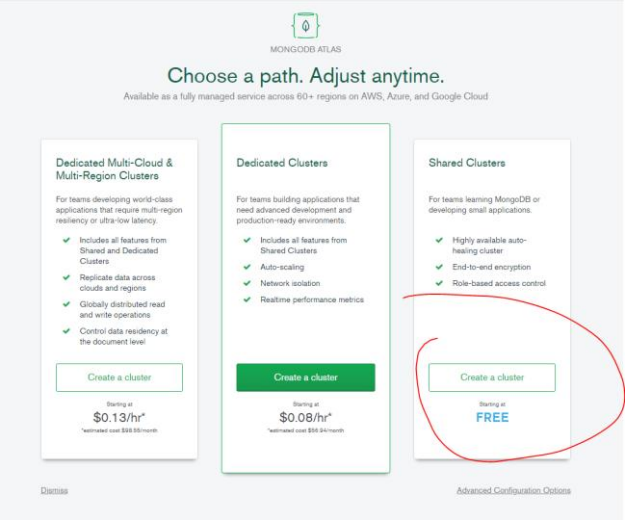
Exercício 2 | Install and configure Atlas + MongoDB

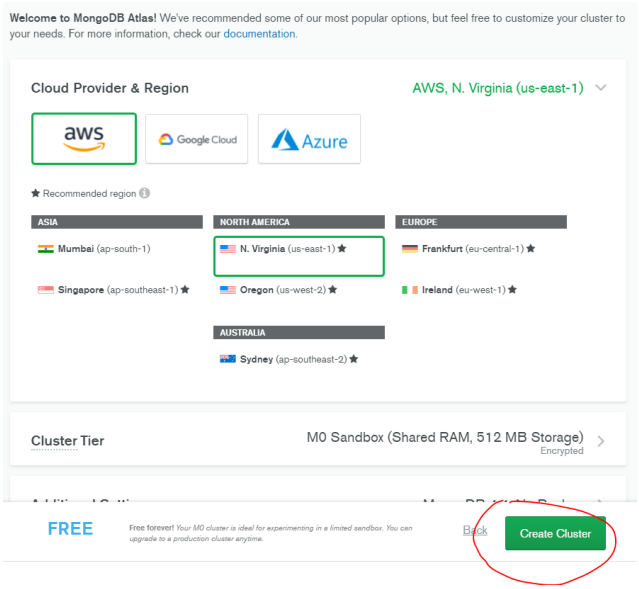
1) Cloud Atlas

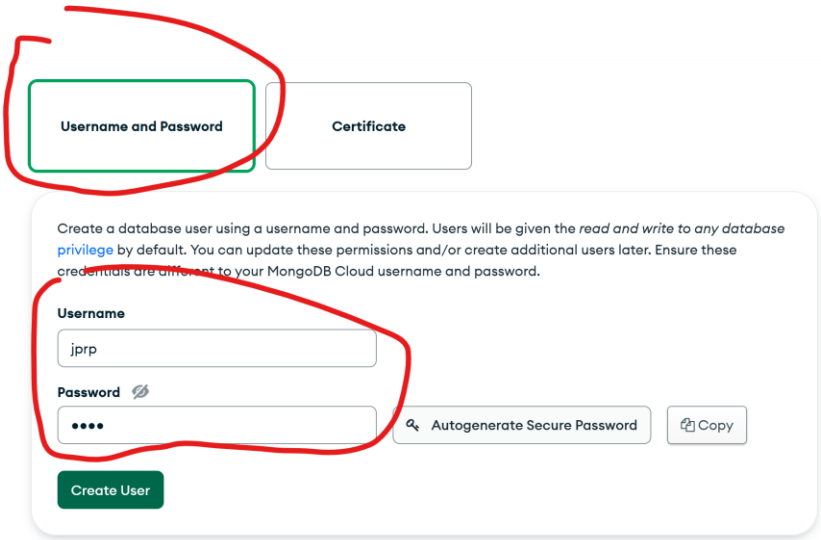
MongoDB Atlas' document model enables developers to store data as JSON-like objects that resemble objects in application code. With MongoDB Atlas, use the tools and languages that you prefer. Manage your clusters with MongoDB CLI.

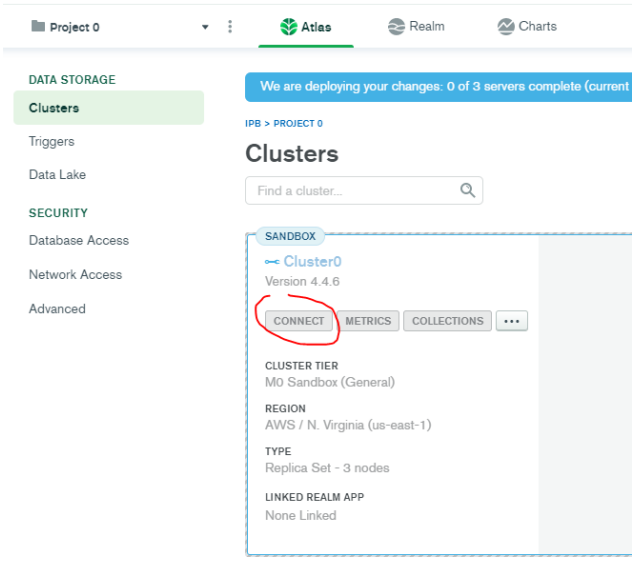
a) Create an account

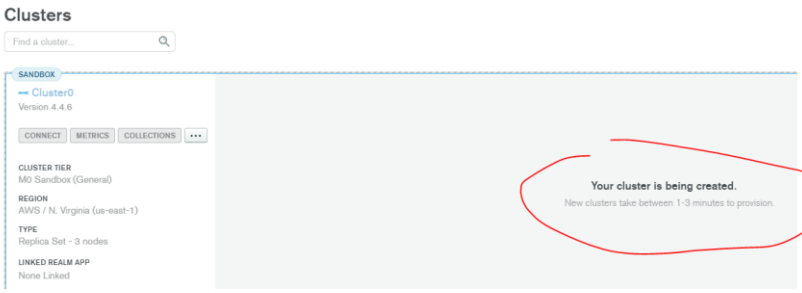
https://www.mongodb.com/cloud/atlas

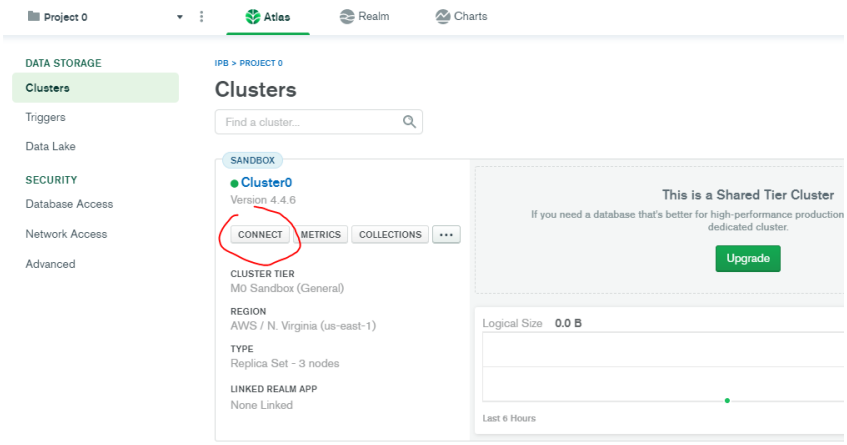


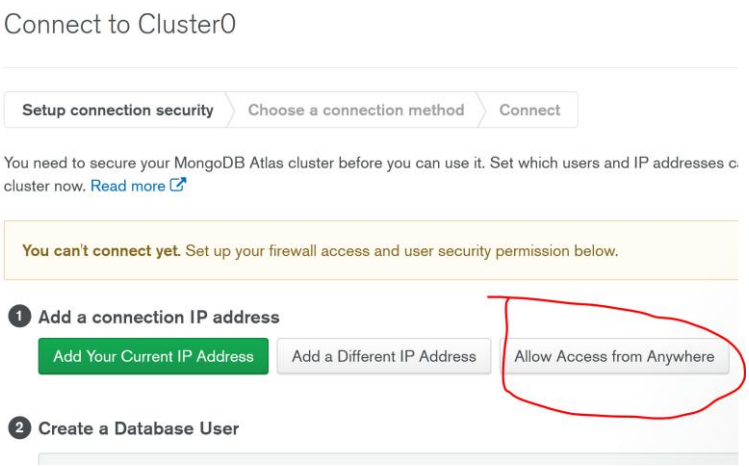


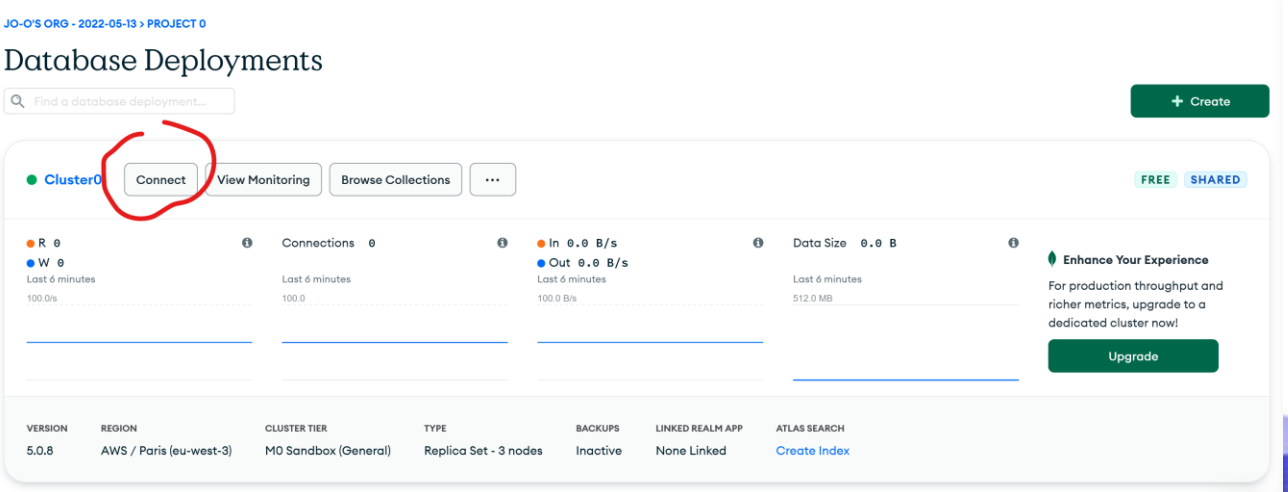

Username and Password
Certificate
Create a database user using a username and password. Users will be given the read and write to any database privilege by default. You can update these permissions and/or create additional users later. Ensure these credentials are different to your MongoDB Cloud username and password.
Username
jprp
Password
••••
Autogenerate Secure Password
Copy
Create User

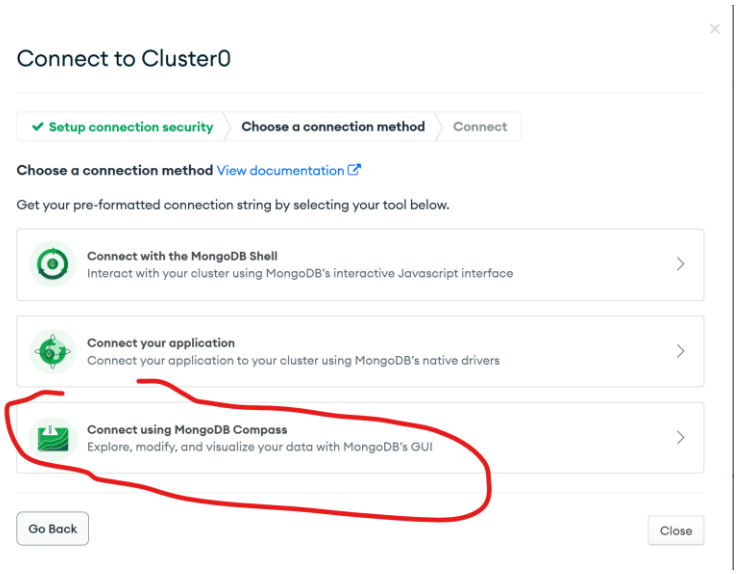

Project 0
Atlas
Realm
Charts
DATA STORAGE
Clusters
Triggers
Data Lake
SECURITY
Database Access
Network Access
Advanced
We are deploying your changes: 0 of 3 servers complete (current
IPB > PROJECT 0
Clusters
Find a cluster...
SANDBOX
Cluster0
Version 4.4.6
CONNECT
METRICS
COLLECTIONS
CLUSTER TIER
M0 Sandbox (General)
REGION
AWS / N. Virginia (us-east-1)
TYPE
Replica Set - 3 nodes
LINKED REALM APP
None Linked

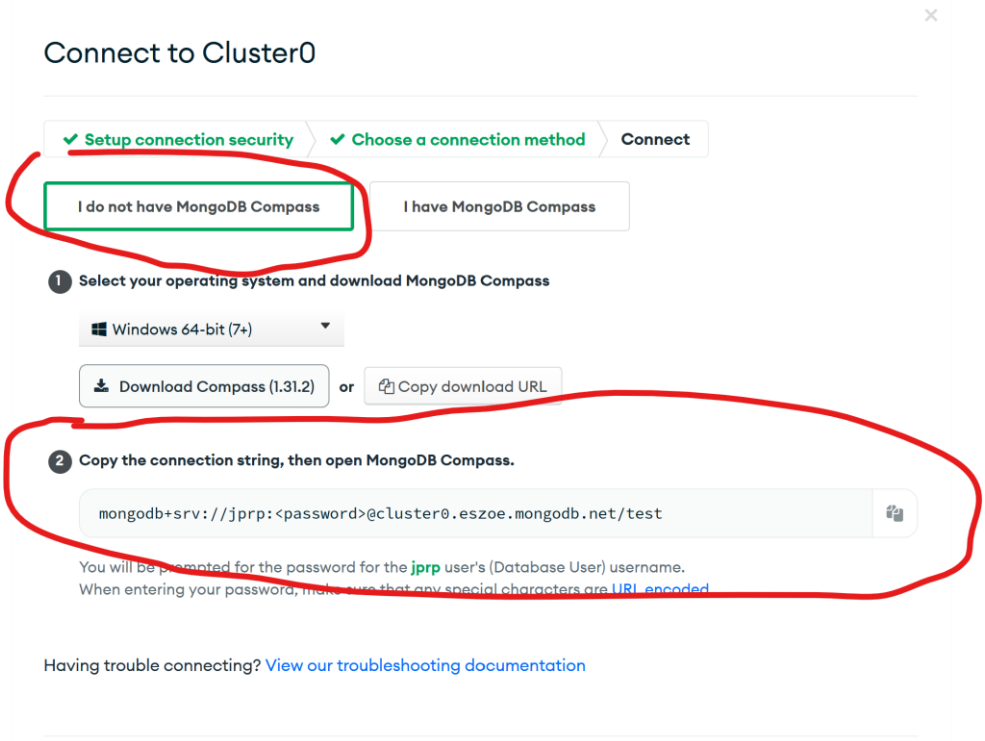

Clusters
Find a cluster...
SANDBOX
Cluster0
Version 4.4.6
CONNECT
METRICS
COLLECTIONS
CLUSTER TIER
M0 Sandbox (General)
REGION
AWS / N. Virginia (us-east-1)
TYPE
Replica Set - 3 nodes
LINKED REALM APP
None Linked
Your cluster is being created.
New clusters take between 1-3 minutes to provision.


Project 0
Atlas
Realm
Charts
DATA STORAGE
Clusters
Triggers
Data Lake
SECURITY
Database Access
Network Access
Advanced
IPB > PROJECT 0
Clusters
Find a cluster...
SANDBOX
Cluster0
Version 4.4.6
CONNECT
METRICS
COLLECTIONS
CLUSTER TIER
M0 Sandbox (General)
REGION
AWS / N. Virginia (us-east-1)
TYPE
Replica Set - 3 nodes
LINKED REALM APP
None Linked
This is a Shared Tier Cluster
If you need a database that's better for high-performance production dedicated cluster.
Upgrade
Logical Size 0.0 B
Last 6 Hours


Connect to Cluster0
Setup connection security
Choose a connection method
Connect
You need to secure your MongoDB Atlas cluster before you can use it. Set which users and IP addresses c
cluster now. Read more
You can't connect yet. Set up your firewall access and user security permission below.
1 Add a connection IP address
Add Your Current IP Address
Add a Different IP Address
Allow Access from Anywhere
2 Create a Database User


JO-O'S ORG - 2022-05-13 > PROJECT 0
Database Deployments
Find a database deployment...
Create
Cluster0
Connect
View Monitoring
Browse Collections
FREE
SHARED
R 0
W 0
Last 6 minutes
Connections 0
In 0.0 B/s
Out 0.0 B/s
Data Size 0.0 B
Enhance Your Experience
For production throughput and richer metrics, upgrade to a dedicated cluster now!
Upgrade
VERSION
5.0.8
REGION
AWS / Paris (eu-west-3)
CLUSTER TIER
M0 Sandbox (General)
TYPE
Replica Set - 3 nodes
BACKUPS
Inactive
LINKED REALM APP
None Linked
ATLAS SEARCH
Create Index



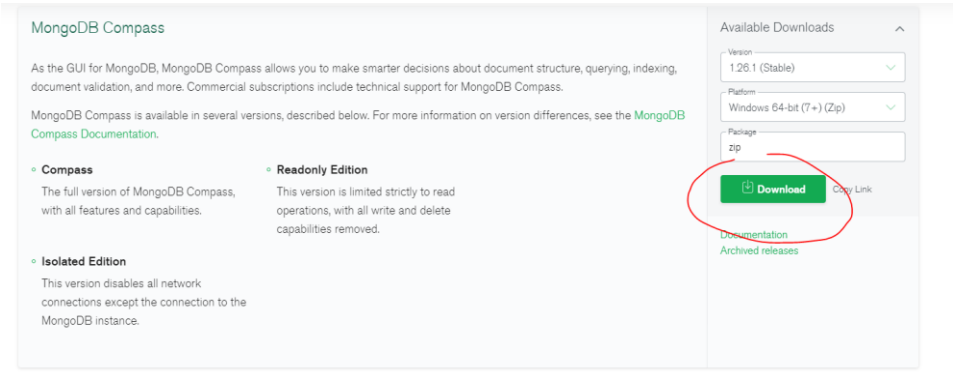


Save the connection string.

2) Install Mongo Compass **(if it is not already installed)**

a) Download Compass

https://www.mongodb.com/try/download/compass?tck=docs_compass



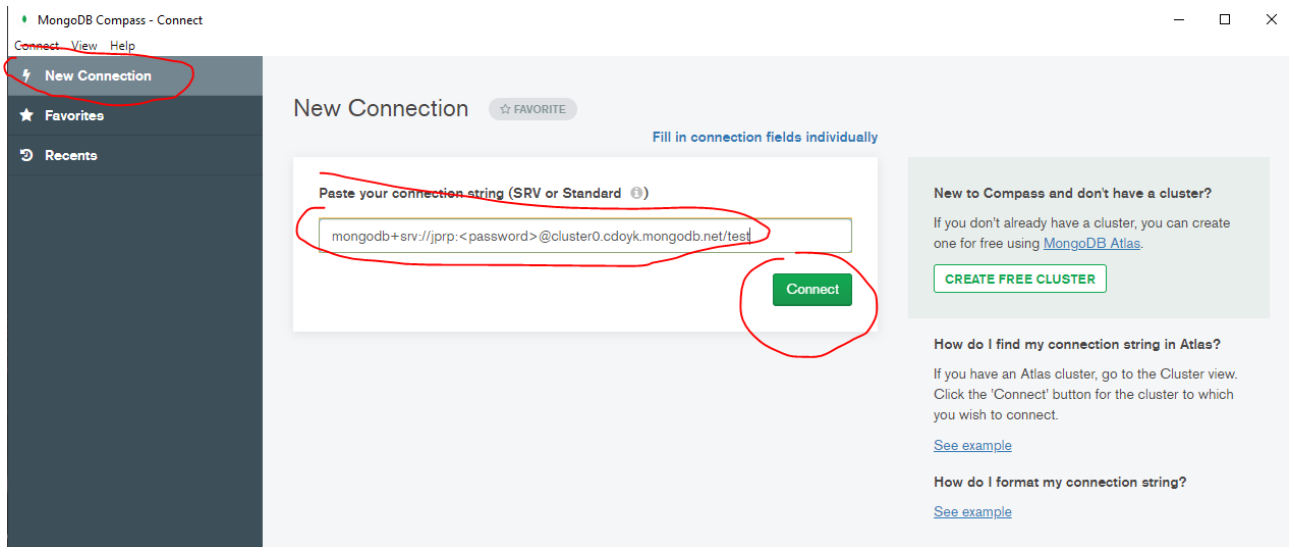
b) Install MongoDB Compass

- Double-click the installer file.
- Follow the instructions to install Compass. You can select the destination of the Compass installation.
- Once installed, Compass launches and prompts you to configure privacy settings and specify update preferences

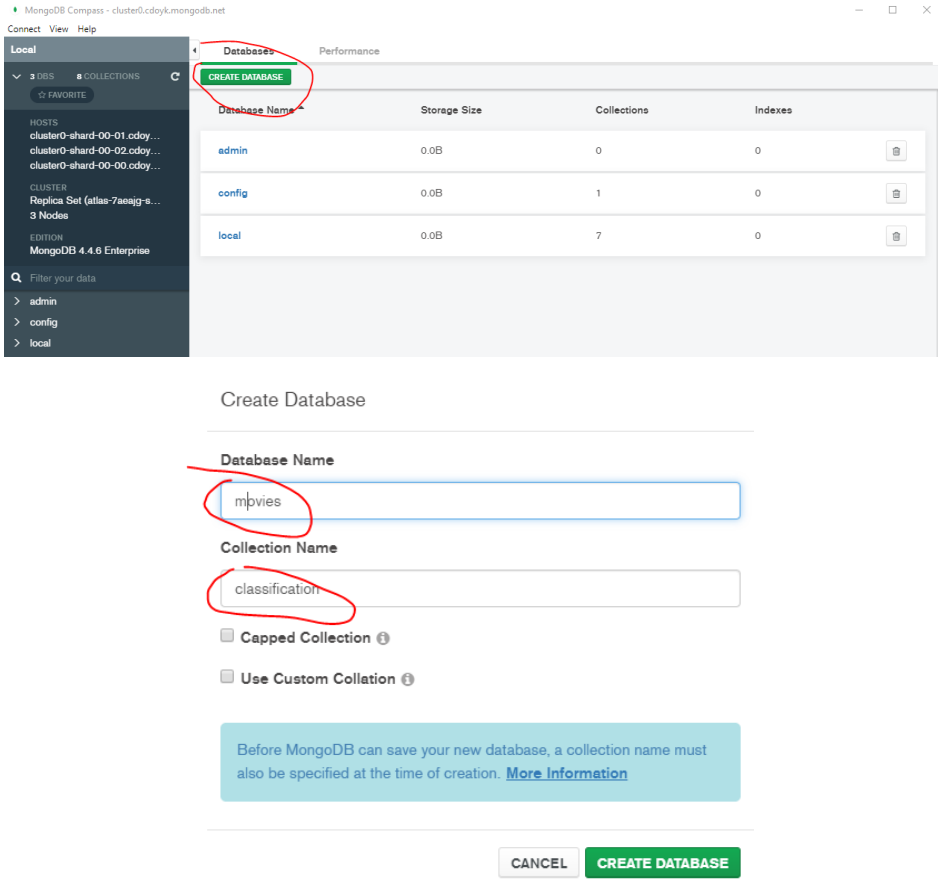
3) Connect MongoDB Compass with the Atlas cloud

https://docs.mongodb.com/compass/master/connect/

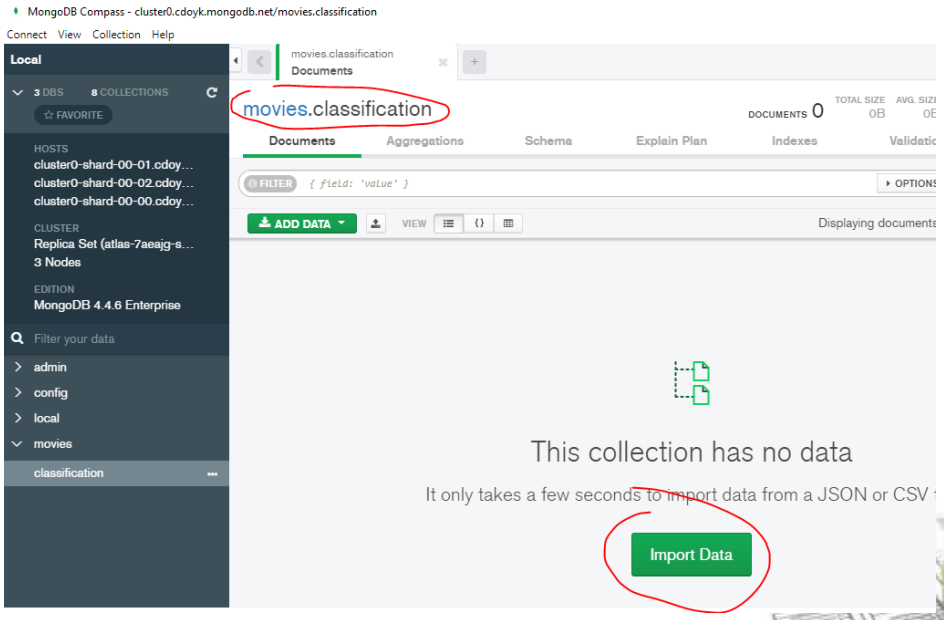
Use the string you saved in your Atlas setup.

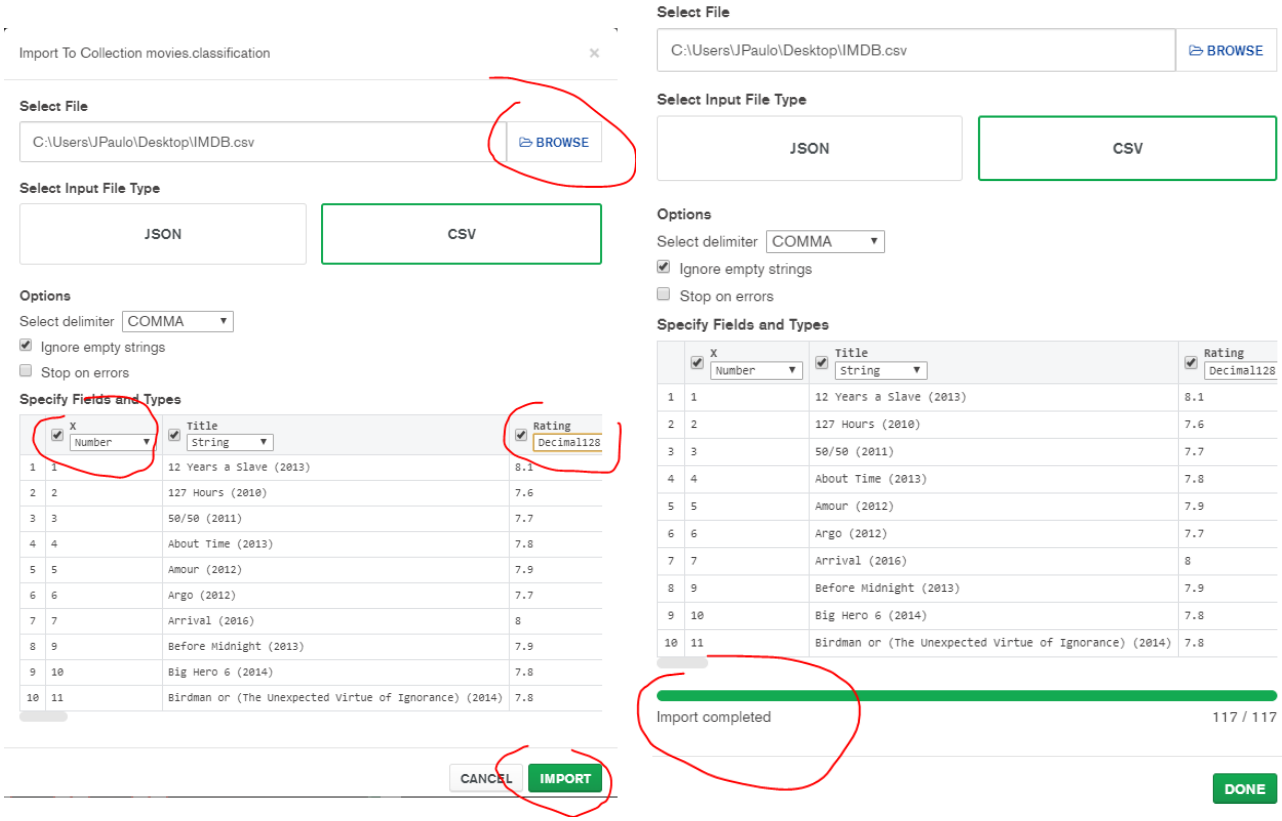


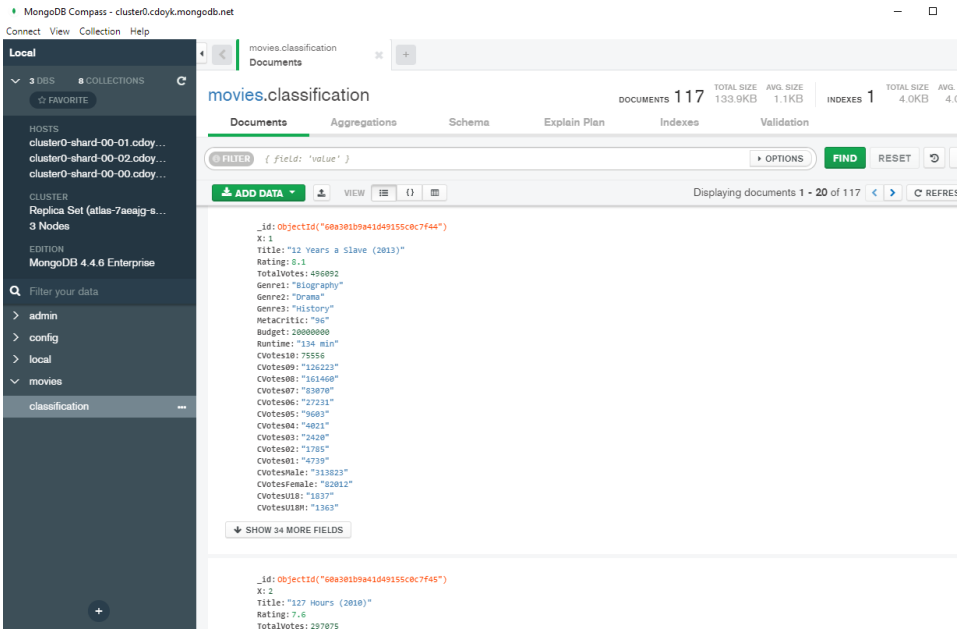
1) Create the "movies" database and the "classification" collection”



2) Import csv

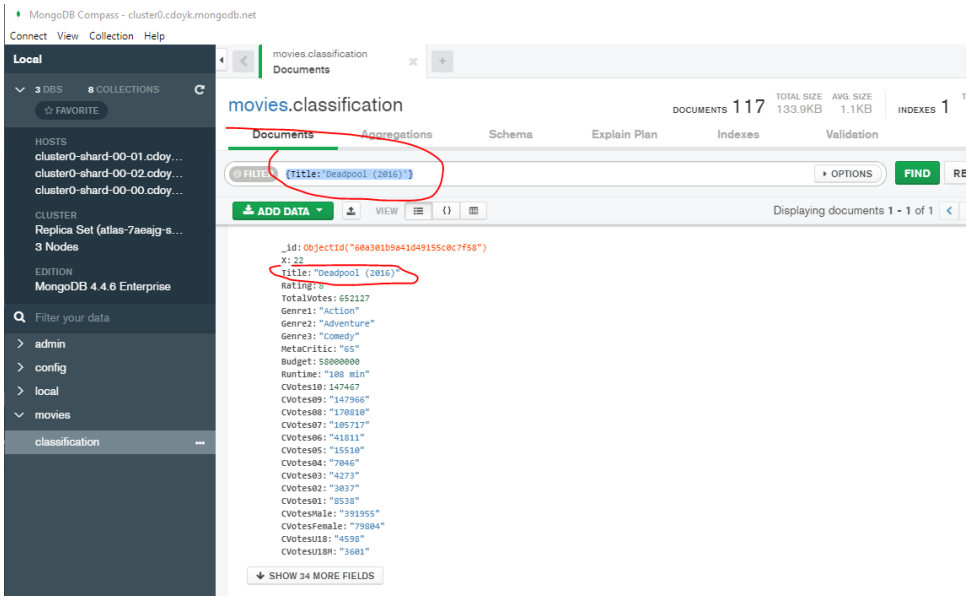




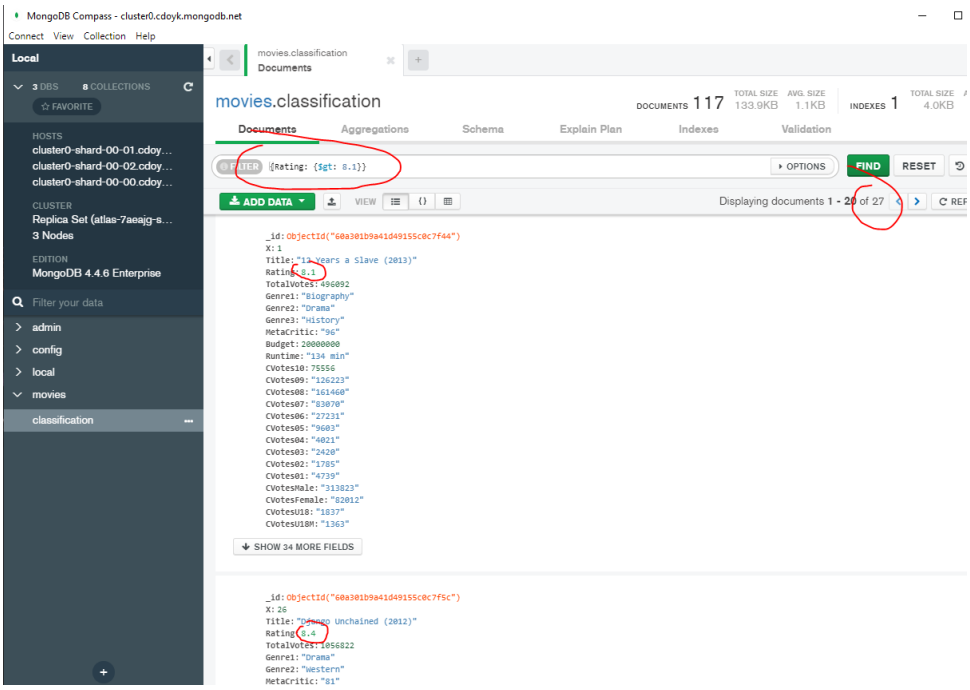


3) Filter data

a) Filter by title 'Deadpool (2016)'

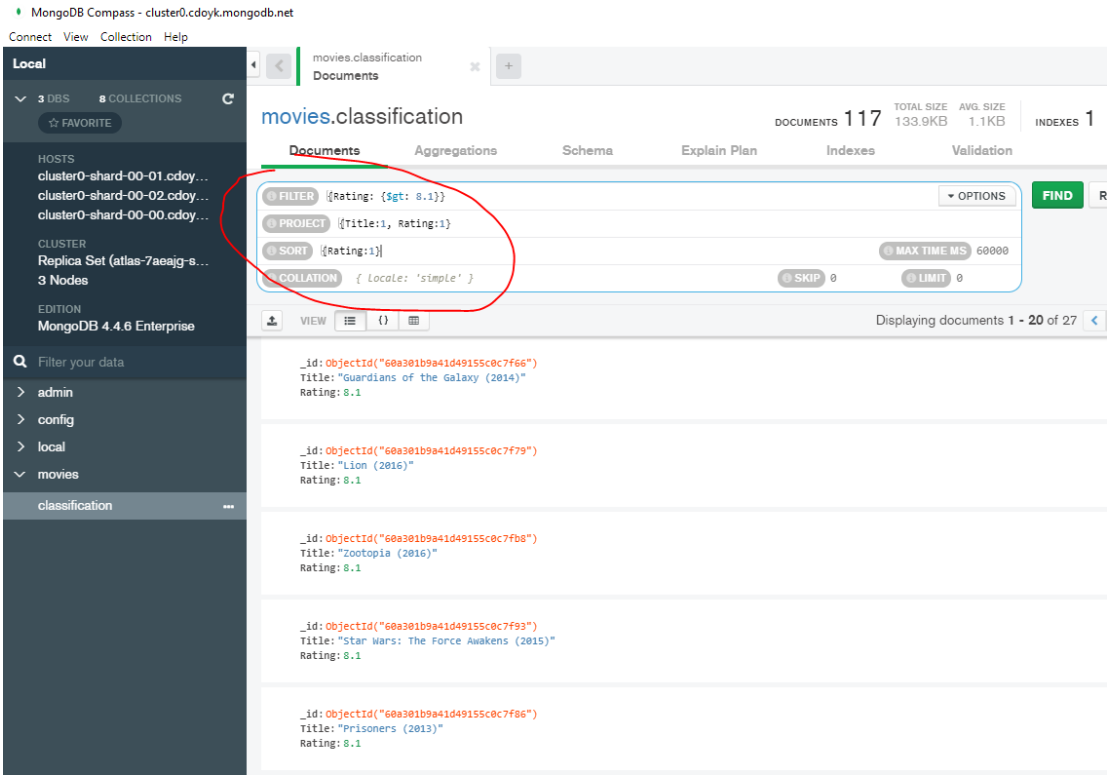


b) Filter all movies with rating > 8.1



c) Filter for all films rated > 8.1 (show title and rating only), and sorted by rating

```
{Rating: {$gt: 8.1}}
{Title:1, Rating:1}
{Rating:1}
```



d) All objects with TotalVotes between 25000 and 30000

```
db.classification.find
(
   { $and:
     [
       { TotalVotes: { $gte: 25000 } },
       { TotalVotes: { $lte: 35000 } }
     ]
   }
)
```

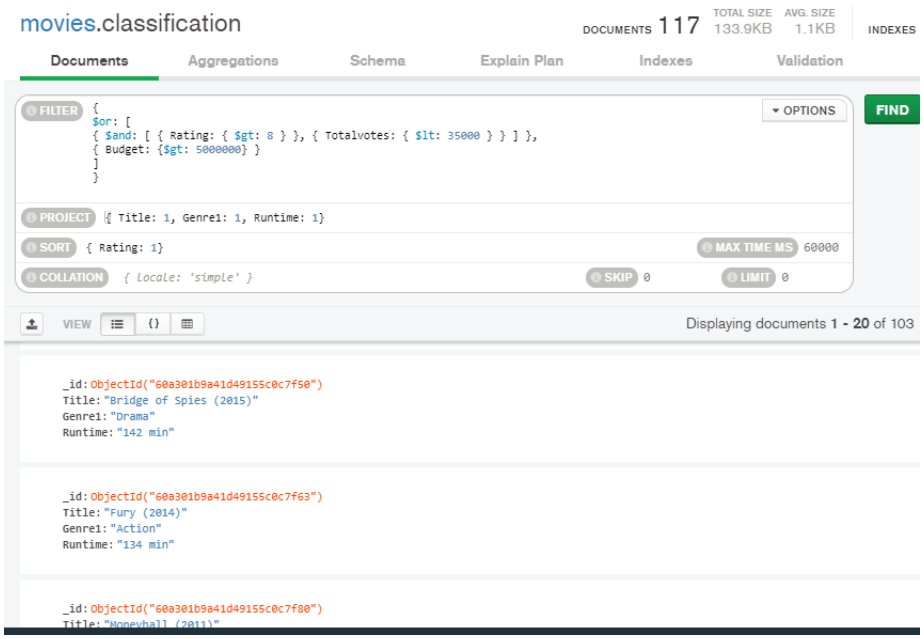
```
db.users.find(
   { age: { $gt: 18 } },
   { name: 1, address: 1 }
).limit(5)
```

← collection
← query criteria
← projection
← cursor modifier

e) Show only Title, Genre1, and Runtime of objects with Rating > 8 and Totalvotes < 35000 or Budget > 5 000 000, sorted by rating (descending).

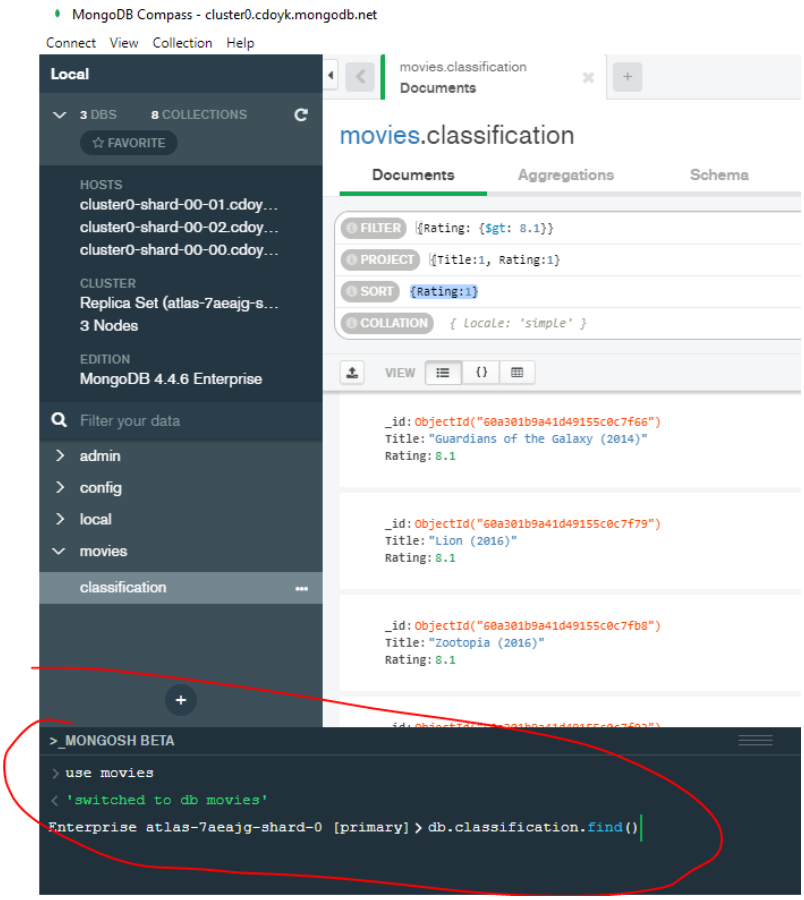
```
{
$or: [
  { $and: [ {Rating: {$gt: 8} }, { Totalvotes: { $lt: 35000 } } ] },
{ Budget: {$gt: 5000000} }
]
}

Project: { Title: 1, Genre1: 1, Runtime: 1}
Sort: { Rating: 1}
```




```
db.classificacao.find
(
    { $or:
    [
            { $and:
                [
                {Rating: {$gt: 8} },
              { Totalvotes: { $lt: 35000 } }
            ]
       },
                { Budget: {$gt: 5000000} }
          ]
    }
)
```

4) Using the SHELL

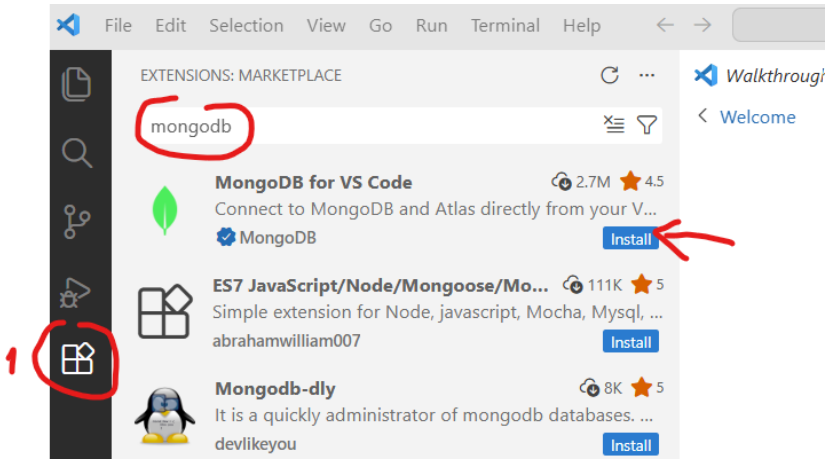



```
>_MONGOSH BETA
> use movies
< 'switched to db movies'
> db.classification.find()
< { _id: ObjectId("60a301b9a41d49155c0c7f44"),
    X: 1,
    Title: '12 Years a Slave (2013)',
    Rating: Decimal128("8.1"),
    TotalVotes: 496092,
    Genre1: 'Biography',
    Genre2: 'Drama',
    Genre3: 'History',
    MetaCritic: '96',
    Budget: 20000000,
    Runtime: '134 min',
    CVotes10: 75556,
    CVotes09: '126223',
```

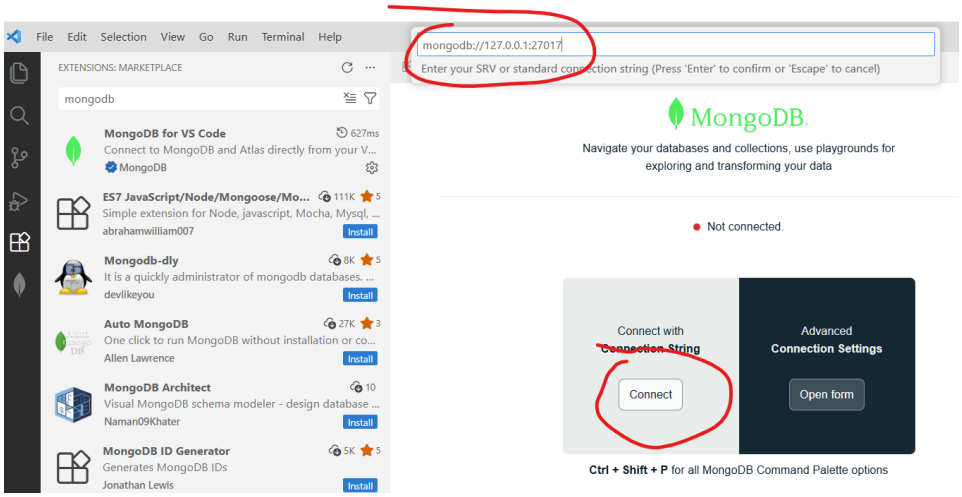
Exercício 4 | Visual Studio code and MongoDB

1) Install MongoDB Extension and connect to DB (local and atlas)

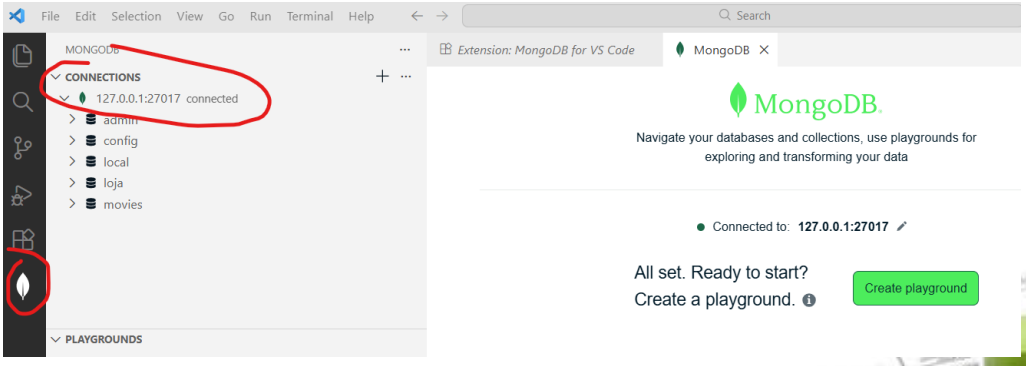
a) Install MongoDB Extension in VS Code



b) Connect to the local server (mongodb://127.0.0.1:27017)



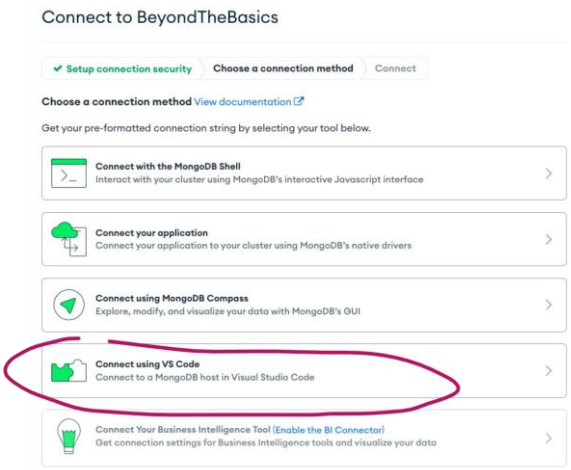
c) Verify the connection



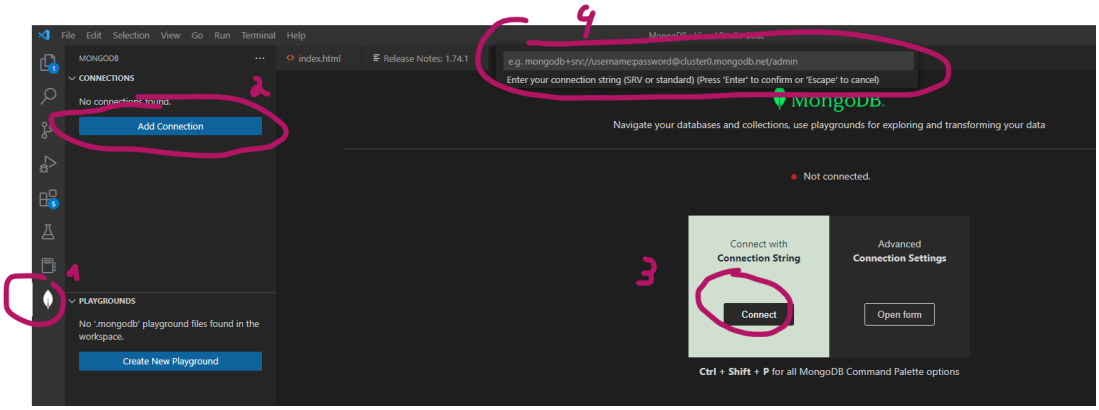
d) Connect to mongoDB Atlas cloud

Use the saved connection string.

Access to MongoDB Atlas

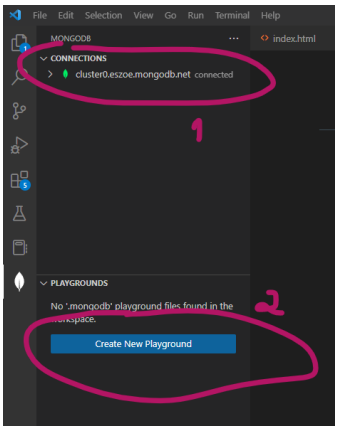


Copy string and paste into VS Code



Example: mongosh "mongodb+srv://jprp:password@cluster0.eszoe.mongodb.net/"

2) Create workspace (New playground)



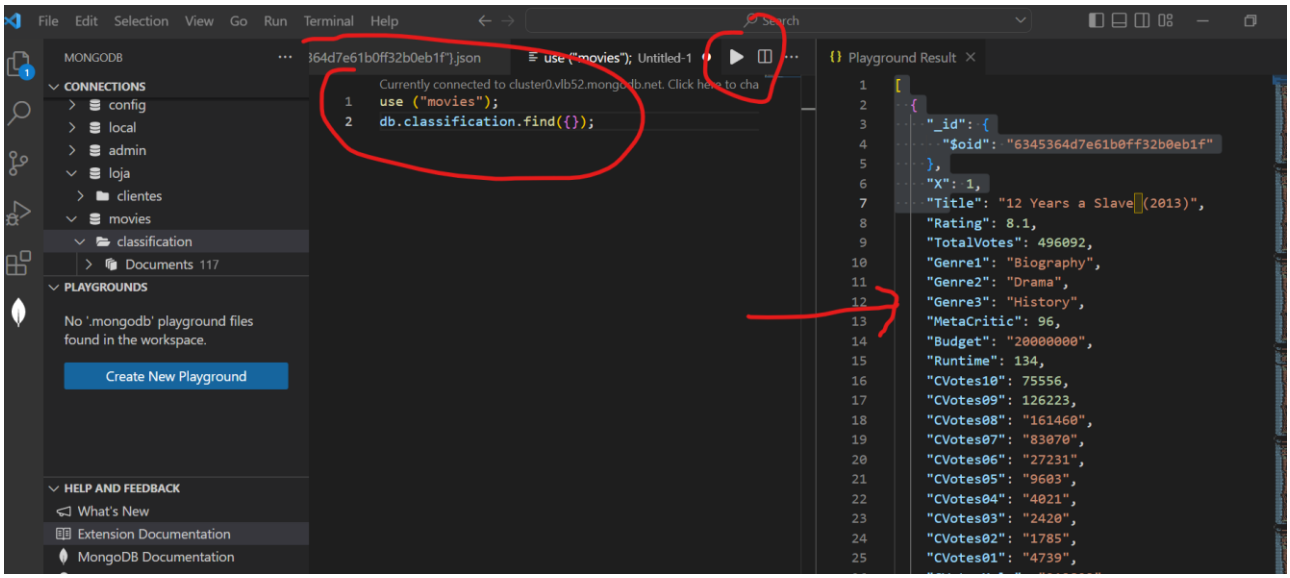
a) Create query

Make a query that allows you to view all the documents in the "classification" collection of the "movies" database.

Query

```
use ("movies");
db.classification.find({});
```

Result



b) Create the "emp" database and the "employees" collection

Create the database and collection

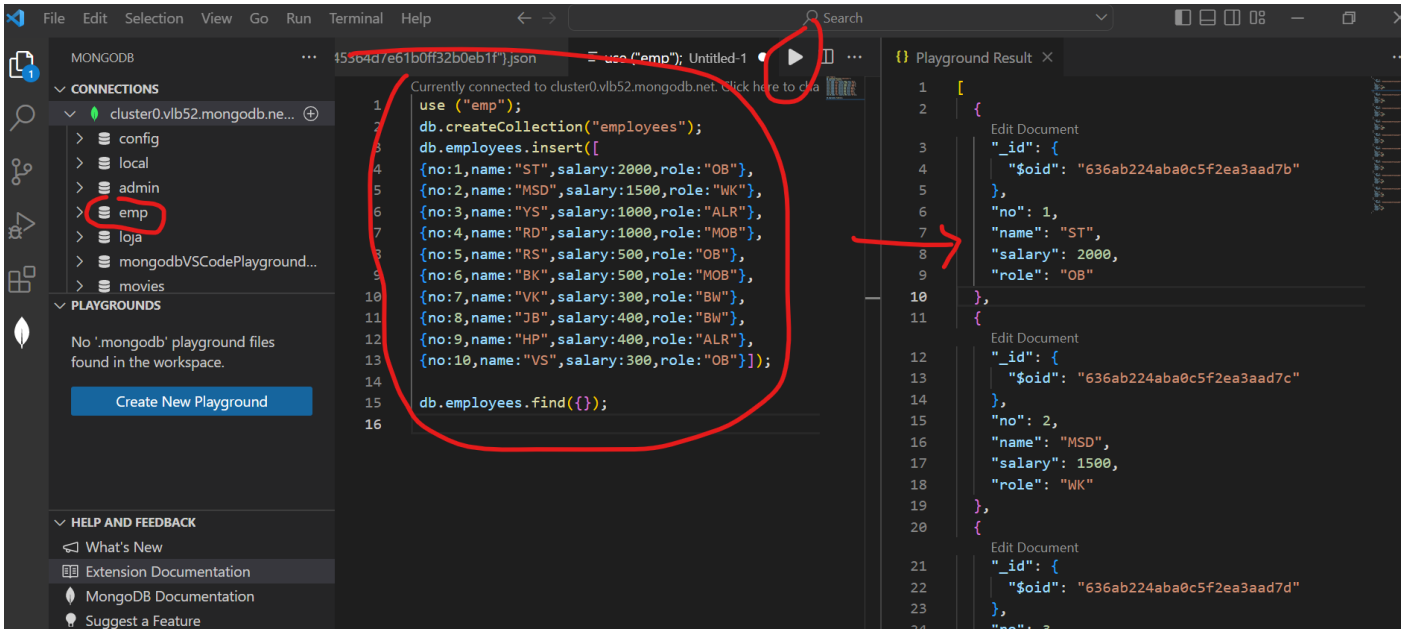
```
use ("emp");
db.createCollection("employees");
```

Insert the following documents:

```
db.employees.insert([
{no:1,name:"ST",salary:2000,role:"OB"},
{no:2,name:"MSD",salary:1500,role:"WK"},
{no:3,name:"YS",salary:1000,role:"ALR"},
{no:4,name:"RD",salary:1000,role:"MOB"},
{no:5,name:"RS",salary:500,role:"OB"},
{no:6,name:"BK",salary:500,role:"MOB"},
{no:7,name:"VK",salary:300,role:"BW"},
{no:8,name:"JB",salary:400,role:"BW"},
{no:9,name:"HP",salary:400,role:"ALR"},
{no:10,name:"VS",salary:300,role:"OB"}]);

db.employees.find({});
```

Result


File Edit Selection View Go Run Terminal Help
MONGODB
CONNECTIONS
cluster0.vlb52.mongodb.ne...
config
local
admin
emp
loja
mongodbVSCodePlayground...
movies
PLAYGROUNDS
No '.mongodb' playground files found in the workspace.
Create New Playground
HELP AND FEEDBACK
What's New
Extension Documentation
MongoDB Documentation
Suggest a Feature
Currently connected to cluster0.vlb52.mongodb.net. Click here to cha
use ("emp"); Untitled-1
use ("emp");
db.createCollection("employees");
db.employees.insert([
{no:1,name:"ST",salary:2000,role:"OB"},
{no:2,name:"MSD",salary:1500,role:"WK"},
{no:3,name:"YS",salary:1000,role:"ALR"},
{no:4,name:"RD",salary:1000,role:"MOB"},
{no:5,name:"RS",salary:500,role:"OB"},
{no:6,name:"BK",salary:500,role:"MOB"},
{no:7,name:"VK",salary:300,role:"BW"},
{no:8,name:"JB",salary:400,role:"BW"},
{no:9,name:"HP",salary:400,role:"ALR"},
{no:10,name:"VS",salary:300,role:"OB"}]);
db.employees.find({});
Playground Result
[
{
Edit Document
"_id": {
"$oid": "636ab224aba0c5f2ea3aad7b"
},
"no": 1,
"name": "ST",
"salary": 2000,
"role": "OB"
},
{
Edit Document
"_id": {
"$oid": "636ab224aba0c5f2ea3aad7c"
},
"no": 2,
"name": "MSD",
"salary": 1500,
"role": "WK"
},
{
Edit Document
"_id": {
"$oid": "636ab224aba0c5f2ea3aad7d"
},

1) Show all available databases in MongoDB.

```
show dbs
```

2) Create the database with the name “Loja”.

```
use loja
```

```
< 'switched to db loja'
```

a) Verify that the DB was created

```
db
```

```
< loja
```

3) Create the collection “clientes”.

a) Create the collection

```
db.createCollection('clientes');
```

```
< { ok: 1 }
```

b) Verify that the collection has been created.

```
show collections
```

```
> show collections
< clientes
```

4) Insert document into collection “clientes”: nome “Joana” and sobrenome “Ferreira”.

```
db.clientes.insert(
    {
          nome:"Joana",
          sobrenome:"Ferreira"
    }
               )
```

```
< 'DeprecationWarning: Collection.insert() is deprecated. Use insertOne, insertMany or bulkWrite.'
< { acknowledged: true,
    insertedIds: { '0': ObjectId("60ab6d2f444158c0a8cae5b7") } }
```

a) Verify that the document has been inserted into the collection “clientes”.

```
db.clientes.find();
```

```
{ _id: ObjectId("60ab6d2f444158c0a8cae5b7"),
  nome: 'Joana',
  sobrenome: 'Ferreira' }
```

5) Insert multiple documents into the collection “clientes”.

```
db.clientes.insert
(
  [
    {
       nome:"Joao",
       sobrenome:"Ribeiro"
    },
    {
       nome:"António",
       sobrenome:"Ribeiro"
    },
    {
       nome:"Vera",
       sobrenome:"Ferreira",
       genero:"Feminino"
    }
  ]
)
```

```
> db.clientes.insert([
        {
           nome:"Joao",
           sobrenome:"Ribeiro"
        },
        {
           nome:"António",
           sobrenome:"Ribeiro"
        },
        {
           nome:"Vera",
           sobrenome:"Ferreira",
           género:"Feminino"
        }
                    ])
{ acknowledged: true,
  insertedIds:
   { '0': ObjectId("60ab6f6a444158c0a8cae5b8"),
     '1': ObjectId("60ab6f6a444158c0a8cae5b9"),
     '2': ObjectId("60ab6f6a444158c0a8cae5ba") } }
```

a) Verify that documents have been inserted into the collection “clientes”.

```
db.clientes.find().pretty()
```

```
> db.clientes.find().pretty()
< { _id: ObjectId("60ab6d2f444158c0a8cae5b7"),
    nome: 'Joana',
    sobrenome: 'Ferreira' }
  { _id: ObjectId("60ab6f6a444158c0a8cae5b8"),
    nome: 'Joao',
    sobrenome: 'Ribeiro' }
  { _id: ObjectId("60ab6f6a444158c0a8cae5b9"),
    nome: 'António',
    sobrenome: 'Ribeiro' }
  { _id: ObjectId("60ab7235444158c0a8cae5bc"),
    nome: 'Vera',
    sobrenome: 'Ferreira',
    genero: 'Feminino' }
```

6) Add a new field ("genero") in the document where the name is "Joao" (genero:"Masculino").

```
db.clientes.update
(
    {nome:"Joao"},
    {$set:{genero:"Masculino"}}
)
```

```
< 'DeprecationWarning: Collection.update() is deprecated. Use updateOne, updateMany or bulkWrite.'
< { acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0 }
> db.clientes.find().pretty()
< { _id: ObjectId("60ab6d2f444158c0a8cae5b7"),
    nome: 'Joana',
    sobrenome: 'Ferreira' }
  { _id: ObjectId("60ab6f6a444158c0a8cae5b8"),
    nome: 'Joao',
    sobrenome: 'Ribeiro',
    genero: 'Masculino' }
  { _id: ObjectId("60ab6f6a444158c0a8cae5b9"),
    nome: 'António',
    sobrenome: 'Ribeiro' }
  { _id: ObjectId("60ab7235444158c0a8cae5bc"),
    nome: 'Vera',
    sobrenome: 'Ferreira',
    genero: 'Feminino' }
```

7) Change the last name of the customer whose first name is "Joao" to "Pereira".

```
db.clientes.update
(
    {nome:"Joao"},
    {$set:{sobrenome:"Pereira"}}
)
```

```
< { acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0 }
> db.clientes.find().pretty()
< { _id: ObjectId("60ab6d2f444158c0a8cae5b7"),
    nome: 'Joana',
    sobrenome: 'Ferreira' }
  { _id: ObjectId("60ab6f6a444158c0a8cae5b8"),
    nome: 'Joao',
    sobrenome: 'Pereira',
    genero: 'Masculino' }
  { _id: ObjectId("60ab6f6a444158c0a8cae5b9"),
    nome: 'António',
    sobrenome: 'Ribeiro' }
  { _id: ObjectId("60ab7235444158c0a8cae5bc"),
    nome: 'Vera',
    sobrenome: 'Ferreira',
    genero: 'Feminino' }
```

8) Delete all documents whose name is “Joao”.

```
db.clientes.remove({nome:"Joao"})
```

```
< 'DeprecationWarning: Collection.remove() is deprecated. Use deleteOne, deleteMany or bulkWrite.'
< { acknowledged: true, deletedCount: 1 }
> db.clientes.find().pretty()
< { _id: ObjectId("60ab6d2f444158c0a8cae5b7"),
    nome: 'Joana',
    sobrenome: 'Ferreira' }
  { _id: ObjectId("60ab6f6a444158c0a8cae5b9"),
    nome: 'António',
    sobrenome: 'Ribeiro' }
  { _id: ObjectId("60ab7235444158c0a8cae5bc"),
    nome: 'Vera',
    sobrenome: 'Ferreira',
    genero: 'Feminino' }
```

a) If we want to delete only the 1st document whose name is “Joao” we can use:

```
db.clientes.remove
(
  {nome:"Joao"},
  {justOne:true}
)
```

9) Insert the following documents into the collection “clientes”:

```
> db.clientes.insert([
    {nome:"Manuel", sobrenome:"Figueira", genero:"Masculino"},
    {nome:"Joaquim", sobrenome:"Pires", genero:"Masculino"},
    {nome:"Maria", sobrenome:"Joaquina", genero:"Feminino"},
    {nome:"João", sobrenome:"Esteves", genero:"Masculino"},
    {nome:"Cristina", sobrenome:"Oliveira", genero:"Feminino"}
                        ])
< { acknowledged: true,
    insertedIds:
     { '0': ObjectId("60ab7698444158c0a8cae5bd"),
       '1': ObjectId("60ab7698444158c0a8cae5be"),
       '2': ObjectId("60ab7698444158c0a8cae5bf"),
       '3': ObjectId("60ab7698444158c0a8cae5c0"),
       '4': ObjectId("60ab7698444158c0a8cae5c1") } }
```

10) View all documents where the name is “Manuel”.

```
db.clientes.find({nome:"Manuel"})
```

```
< { _id: ObjectId("60ab7698444158c0a8cae5bd"),
    nome: 'Manuel',
    sobrenome: 'Figueira',
    genero: 'Masculino' }
```

11) Search all results that contain the name “João” or the gender is “Feminino”.

```
db.clientes.find
(
    {$or:
        [
          {nome:"João"},
          {genero:"Feminino"}
        ]
    }
)
```

```
{ _id: ObjectId("60ab7235444158c0a8cae5bc"),
  nome: 'Vera',
  sobrenome: 'Ferreira',
  genero: 'Feminino' }
{ _id: ObjectId("60ab7698444158c0a8cae5bf"),
  nome: 'Maria',
  sobrenome: 'Joaquina',
  genero: 'Feminino' }
{ _id: ObjectId("60ab7698444158c0a8cae5c0"),
  nome: 'João',
  sobrenome: 'Esteves',
  genero: 'Masculino' }
{ _id: ObjectId("60ab7698444158c0a8cae5c1"),
  nome: 'Cristina',
  sobrenome: 'Oliveira',
  genero: 'Feminino' }
```

12) Show all documents sorted alphabetically by field “nome”.

```
db.clientes.find().sort({nome:1})
```

13) Count the documents of customers of this kind “Feminino”

```
db.clientes.find
(
   {genero:"Feminino"}
).count()
```

```
3
```

11) Add a new field (“estado”) to the document with a name "João" (estado: “Suspenso”).

```
db.clientes.update
(
  {nome:"João"},
  {$set:{estado:"Suspenso"}}
)
```

```
{ acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0 }
db.clientes.find({nome:"João"})
{ _id: ObjectId("60ab7698444158c0a8cae5c0"),
  nome: 'João',
  sobrenome: 'Esteves',
  genero: 'Masculino',
  estado: 'Suspenso' }
```

14) Write a query in MongoDB that shows the fields “nome” and “genero” of all documents in the collection “clientes”.

```
db.clientes.find
(
   {},
   {"nome" : 1, "genero":1}
);
```

Exercício 6 | Employees

Create the database “EMP” and a collection with the name "EMPLOYEES".

1) Create the "emp" Database

```
use emp
```

2) Create the collection EMPLOYEES

```
db.createCollection("employees")
```

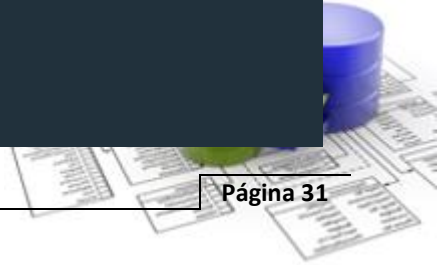
```
< 'switched to db emp'
> db
< emp
> db.createCollection("employees")
< { ok: 1 }
```

3) Insert the following records into the collection “employees”

```
db.employees.insert([
 {no:1,name:"ST",salary:2000,role:"OB"},
 {no:2,name:"MSD",salary:1500,role:"WK"},
 {no:3,name:"YS",salary:1000,role:"ALR"},
 {no:4,name:"RD",salary:1000,role:"MOB"},
 {no:5,name:"RS",salary:500,role:"OB"},
 {no:6,name:"BK",salary:500,role:"MOB"},
 {no:7,name:"VK",salary:300,role:"BW"},
 {no:8,name:"JB",salary:400,role:"BW"},
 {no:9,name:"HP",salary:400,role:"ALR"},
 {no:10,name:"VS",salary:300,role:"OB"}])
```

```
db.users.insertOne(          ← collection
  {
    name: "sue",             ← field: value
    age: 26,                 ← field: value
    status: "pending"        ← field: value
  }                                          } document
)
```

```
> db.employees.insert([
  {no:1,name:"ST",salary:2000,role:"OB"},
  {no:2,name:"MSD",salary:1500,role:"WK"},
  {no:3,name:"YS",salary:1000,role:"ALR"},
  {no:4,name:"RD",salary:1000,role:"MOB"},
  {no:5,name:"RS",salary:500,role:"OB"},
  {no:6,name:"BK",salary:500,role:"MOB"},
  {no:7,name:"VK",salary:300,role:"BW"},
  {no:8,name:"JB",salary:400,role:"BW"},
  {no:9,name:"HP",salary:400,role:"ALR"},
  {no:10,name:"VS",salary:300,role:"OB"}])
< 'DeprecationWarning: Collection.insert() is deprecated. Use insertOne, insertMany or bulkWrite.'
< { acknowledged: true,
   insertedIds:
    { '0': ObjectId("60a30a2dad3c1fc590f91a8d"),
      '1': ObjectId("60a30a2dad3c1fc590f91a8e"),
      '2': ObjectId("60a30a2dad3c1fc590f91a8f"),
      '3': ObjectId("60a30a2dad3c1fc590f91a90"),
      '4': ObjectId("60a30a2dad3c1fc590f91a91"),
      '5': ObjectId("60a30a2dad3c1fc590f91a92"),
      '6': ObjectId("60a30a2dad3c1fc590f91a93"),
      '7': ObjectId("60a30a2dad3c1fc590f91a94"),
      '8': ObjectId("60a30a2dad3c1fc590f91a95"),
      '9': ObjectId("60a30a2dad3c1fc590f91a96") } }
```



4) Show all documents in the collection “employees” .

```
db.employees.find().pretty()
```

```
{ _id: ObjectId("60a30a2dad3c1fc590f91a8d"),
  no: 1,
  name: 'ST',
  salary: 2000,
  role: 'OB' }
{ _id: ObjectId("60a30a2dad3c1fc590f91a8e"),
  no: 2,
  name: 'MSD',
  salary: 1500,
  role: 'WK' }
{ _id: ObjectId("60a30a2dad3c1fc590f91a8f"),
  no: 3,
  name: 'YS',
  salary: 1000,
  role: 'ALR' }
{ _id: ObjectId("60a30a2dad3c1fc590f91a90"),
  no: 4,
  name: 'RD',
  salary: 1000,
  role: 'MOB' }
{ _id: ObjectId("60a30a2dad3c1fc590f91a91"),
  no: 5,
```

5) Update the salary of the employee with name "ST" with an increase of +8000.

```
db.employees.update
(
    {name:"ST"},
    {$inc:{salary:8000}}
  }
)
```

```
'DeprecationWarning: Collection.update() is deprecated. Use updateOne, updateMany or bulkWrite.'
{ acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0 }
db.employees.find().pretty()
{ _id: ObjectId("60a30a2dad3c1fc590f91a8d"),
  no: 1,
  name: 'ST',
  salary: 10000,
  role: 'OB' }
{ _id: ObjectId("60a30a2dad3c1fc590f91a8e"),
  no: 2,
  name: 'MSD',
```

6) Update the "Salary" field of all employees with an increase of +4000.

```
db.employees.update({},{$inc:{salary:4000}},{multi:true})
```

```
< 'DeprecationWarning: Collection.update() is deprecated. Use updateOne, updateMany or bulkWrite.'
< { acknowledged: true,
    insertedId: null,
    matchedCount: 10,
    modifiedCount: 10,
    upsertedCount: 0 }
```

```
> db.employees.find().pretty()
< { _id: ObjectId("60a30a2dad3c1fc590f91a8d"),
    no: 1,
    name: 'ST',
    salary: 14000,
    role: 'OB' }
  { _id: ObjectId("60a30a2dad3c1fc590f91a8e"),
    no: 2,
    name: 'MSD',
    salary: 5500,
```

7) Update the "ROLE" field from theemplyee "MSD" for "C and WK".

```
db.employees.update({name:"MSD"},{$set:{role:"c and WK"}})
```

```
< { acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0 }
```

```
> db.employees.find({name:'MSD'}).pretty()
< { _id: ObjectId("60a30a2dad3c1fc590f91a8e"),
    no: 2,
    name: 'MSD',
    salary: 5500,
    role: 'c and WK' }
```

8) Add a new field “remark” to the document with a name "RS" (remark: “WC”).

```
db.employees.update({name:"RS"},{$set:{remark:"WC"}})
```

```
< 'DeprecationWarning: Collection.update() is deprecated. Use updateOne, updateMany or bulkWrite.'
< { acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0 }
```

```
> db.employees.find({name:'RS'})
< { _id: ObjectId("60a30a2dad3c1fc590f91a91"),
    no: 5,
    name: 'RS',
    salary: 4500,
    role: 'OB',
    remark: 'WC' }
```

9) Add a new record with number 11, name “AK”, role “coch”, and salary 10000.

```
db.employees.insert
(
  [
    {no:11,
    name:"AK",
    salary:10000,
    role:"coch"},
  ]
)
```

```
> db.employees.find({no:11})
< { _id: ObjectId("60a413c3e6c733ba99444de8"),
    no: 11,
    name: 'AK',
    role: 'coch',
    salary: 10000 }
```

10) Delete the record added in the previous point.

```
db.employees.deleteMany(
    { no: 11 }
)
```

```
db.users.deleteMany(          <-- collection
  { status: "reject" }        <-- delete filter
)
```

```
> db.employees.deleteMany({ no: 11 })
< { acknowledged: true, deletedCount: 1 }
```

or

```
db. employees.remove({ no: 11 })
```

11) Add (and remove) a new "remark" field to the document with the name "RD" (remark: “WC”).

a) Add "remark" field”

```
db.employees.update
(
    {name:"RD"},
    {$set:{remark:"WC"}}
)
```

a) Delete "remark" field”

```
db.employees.update({name:"RD"},{$unset:{remark:"WC"}})
```

```
> db.employees.find({name:'RD'})
< { _id: ObjectId("60a30a2dad3c1fc590f91a90"),
    no: 4,
    name: 'RD',
    salary: 5000,
    role: 'MOB',
    remark: 'WC' }
```

```
> db.employees.update({name:"RD"},{$unset:{remark:"WC"}})
< { acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0 }
> db.employees.find({name:'RD'})
< { _id: ObjectId("60a30a2dad3c1fc590f91a90"),
    no: 4,
    name: 'RD',
    salary: 5000,
    role: 'MOB' }
```

12) Update the document whose "name" field is "RD", by multiplying the salary by 2.

```
db.employees.update({name:"RD"},{$mul:{salary:2}})
```

```
> db.employees.update({name:"RD"},{$mul:{salary:2}})
< { acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0 }
```

```
> db.employees.find({name:'RD'})
< { _id: ObjectId("60a30a2dad3c1fc590f91a90"),
    no: 4,
    name: 'RD',
    salary: 10000,
    role: 'MOB' }
```

13) Find the documents in the EMPLOYEES collection where the name begins with “S”.

```
db.employees.find({name:/^S/})
```

14) Find the documents in the EMPLOYEES collection where the name ends in “K”.

```
db.employees.find({name:/K$/})
```

15) Find the documents in the EMPLOYEES collection where the name has "S" in any position.

```
db.employees.find({name:/S/})
```

16) Show documents in the EMPLOYEES collection where the role is "OB" and "MOB".

```
db.employees.find({role:{$in:["OB","MOB"]}})
```

> Use of $in and $nin (in and notin)
> Note: There will not use {} braces in that $in and $nin

17) Show documents in the EMPLOYEES collection where the role is not "OB" and "MOB".

```
db.employees.find({role:{$nin:["OB","MOB"]}})
```

Exercício 7 | Restaurant

1) Create the database “REST” and a collection with the name " RESTAURANTS ".

a) Create the Database “rest”.

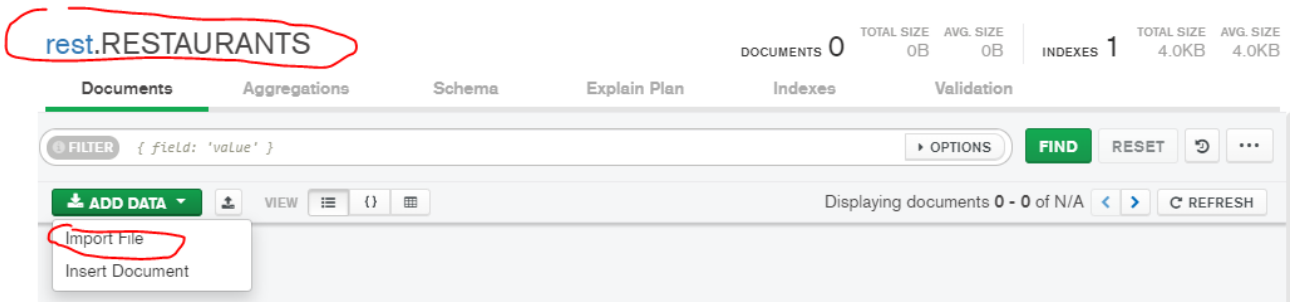
b) Create the collection “RESTAURANTS”.

2) Import data from the file “restaurants.json” to the collection “RESTAURANTS”.

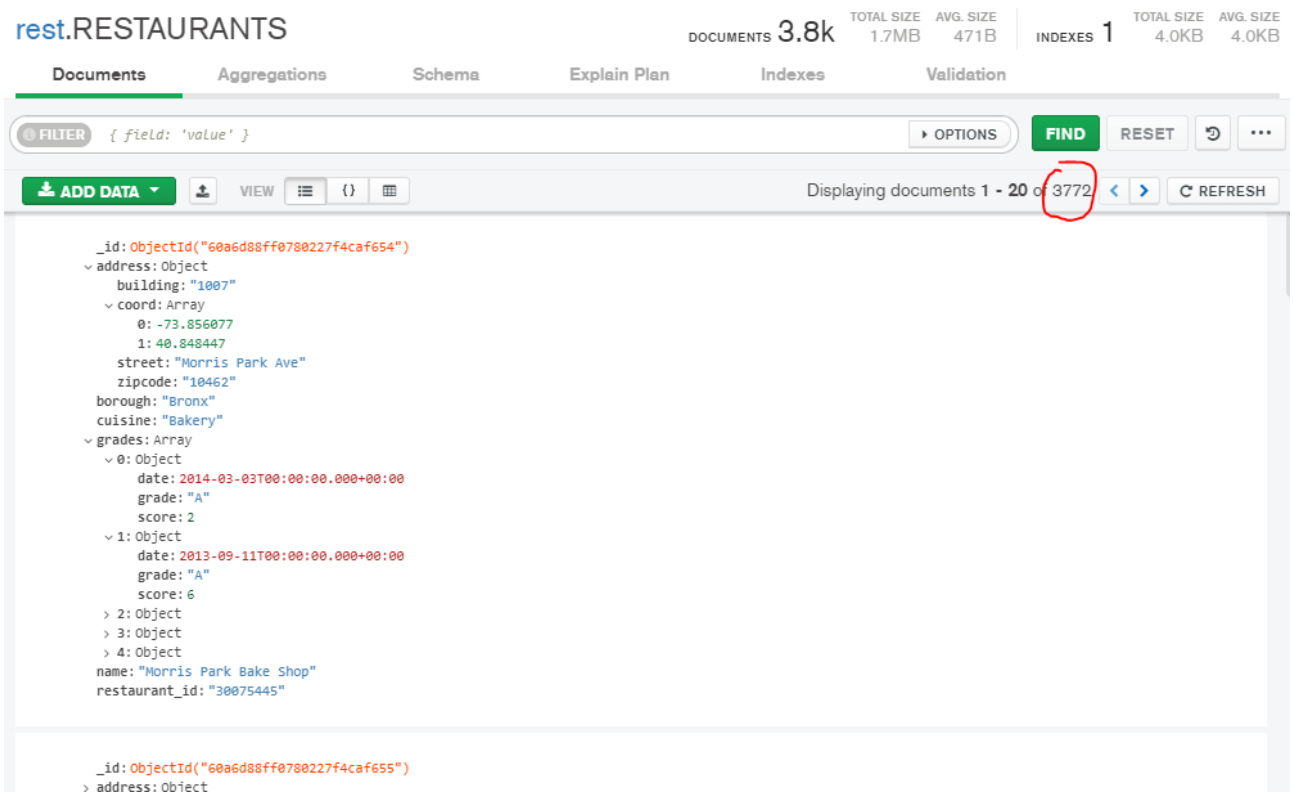
Collection structure “RESTAURANTS”:

```
{
  "address":
    {
      "building": "1007",
      "coord": [ -73.856077, 40.848447 ],
      "street": "Morris Park Ave",
      "zipcode": "10462"
    },
  "borough": "Bronx",
  "cuisine": "Bakery",
  "grades": [
    { "date": { "$date": 1393804800000 }, "grade": "A", "score": 2 },
    { "date": { "$date": 1378857600000 }, "grade": "A", "score": 6 },
    { "date": { "$date": 1358985600000 }, "grade": "A", "score": 10 },
    { "date": { "$date": 1322006400000 }, "grade": "A", "score": 9 },
    { "date": { "$date": 1299715200000 }, "grade": "B", "score": 14 }
  ],
  "name": "Morris Park Bake Shop",
  "restaurant_id": "30075445"
}
```

a) Load data into the DB “REST” Collection “RESTAURANTS”.



Loaded data



MongoDB	Collection structure
_id: ObjectId("60a6d88ff0780227f4caf654") address: Object building: "1007" coord: Array 0: -73.856077 1: 40.848447 street: "Morris Park Ave" zipcode: "10462" borough: "Bronx" cuisine: "Bakery" grades: Array 0: Object date: 2014-03-03T00:00:00.000+00:00 grade: "A" score: 2 1: Object date: 2013-09-11T00:00:00.000+00:00 grade: "A" score: 6 2: Object 3: Object 4: Object name: "Morris Park Bake Shop" restaurant_id: "30075445"	{ "address": { "building": "1007", "coord": [-73.856077, 40.848447], "street": "Morris Park Ave", "zipcode": "10462" }, "borough": "Bronx", "cuisine": "Bakery", "grades": [{ "date": { "$date": 1393804800000 }, "grade": "A", "score": 2 }, { "date": { "$date": 1378857600000 }, "grade": "A", "score": 6 }, { "date": { "$date": 1358985600000 }, "grade": "A", "score": 10 }, { "date": { "$date": 1322006400000 }, "grade": "A", "score": 9 }, { "date": { "$date": 1299715200000 }, "grade": "B", "score": 14 }], "name": "Morris Park Bake Shop", "restaurant_id": "30075445" }

3) Write a query in MongoDB that shows all the documents in the collection RESTAURANTS.

4) Write a query in MongoDB that shows the "restaurant_id", "name", "borough", and "cuisine" fields of all documents in the RESTAURANTS collection.

```
{ _id: ObjectId("60a6d88ff0780227f4caf654"),
  borough: 'Bronx',
  cuisine: 'Bakery',
  name: 'Morris Park Bake Shop',
  restaurant_id: '30075445' }
{ _id: ObjectId("60a6d88ff0780227f4caf655"),
  borough: 'Brooklyn',
  cuisine: 'Hamburgers',
```

5) Write a query in MongoDB that shows the fields "restaurant_id", "name", "borough", and "cuisine", but excludes the field "_id", from all documents in the RESTAURANTS collection

```
{ borough: 'Bronx',
  cuisine: 'Bakery',
  name: 'Morris Park Bake Shop',
  restaurant_id: '30075445' }
{ borough: 'Brooklyn',
  cuisine: 'Hamburgers',
```

6) Write a query in MongoDB that shows the "restaurant_id", "name", "borough", and "zipcode" fields, but excludes the "_id" field from all documents in the RESTAURANTS collection

```
{ address: { zipcode: '10462' },
  borough: 'Bronx',
  name: 'Morris Park Bake Shop',
  restaurant_id: '30075445' }
{ address: { zipcode: '11225' },
  borough: 'Brooklyn',
  name: 'Wendy\'S',
  restaurant_id: '30112340' }
{ address: { zipcode: '10019' },
  borough: 'Manhattan',
```

7) Write a query in MongoDB that shows all restaurants where the "borough" field is "Bronx".

8) Write a query in MongoDB that shows the first 5 restaurants where the "borough" field is "Bronx".

9) Write a query in MongoDB that skips the first 5 restaurants and shows the next 5 where the "borough" field is "Bronx".

10) Write a query in MongoDB that finds restaurants that have a score higher than 90.

11) Write a query in MongoDB that finds restaurants that have a score higher than 80 and less than 100. Fields to show: "name", "borough", and "cuisine".

```
{ _id: ObjectId("60a6d88ff0780227f4caf853"),
  borough: 'Manhattan',
  cuisine: 'Indian',
  name: 'Gandhi' }
{ _id: ObjectId("60a6d88ff0780227f4caf9b6"),
  borough: 'Manhattan',
  cuisine: 'Pizza/Italian',
  name: 'Bella Napoli' }
{ _id: ObjectId("60a6d891f0780227f4cb0222"),
  borough: 'Manhattan',
  cuisine: 'American ',
  name: 'West 79Th Street Boat Basin Cafe' }
```

12) Write a query in MongoDB that finds restaurants that are located at a latitude lower than "-95.754168".

```
_id: ObjectId("60a6d890f0780227f4cafc9c")
address: Object
  building: "3707"
  coord: Array
    0: -101.8945214
    1: 33.5197474
  street: "82 Street"
  zipcode: "11372"
borough: "Queens"
cuisine: "American "
grades: Array
name: "Burger King"
restaurant_id: "40534067"
```

```
_id: ObjectId("60a6d891f0780227f4cb0007")
address: Object
  building: "15259"
  coord: Array
    0: -119.6368672
    1: 36.2504996
  street: "10 Avenue"
```

13) Write a query in MongoDB that finds restaurants that do not prepare "cuisine" of type "American", that the "score" is greater than 70, and the "latitude" is less than -65.754168.

```
db.RESTAURANTS.find(
                {$and:
                      [
                         {"cuisine" : {$ne :"American "}},
                         {"grades.score" : {$gt : 70}},
                         {"address.coord" : {$lt : -65.754168}}
                      ]
                 }
                      )
```

a) Write the query without using the $and operator

```
db.RESTAURANTS.find(
                        {
                          "cuisine" : {$ne : "American "},
                          "grades.score" :{$gt: 70},
                          "address.coord" : {$lt : -65.754168}
                        }
                       )
```

14) Write a query in MongoDB that finds restaurants that do not prepare "American" cuisine, have an "A" rating (grade), and are not located in "Brooklyn" (borough). The document must be submitted in descending order of the "cuisine" field.

15) Write a query in MongoDB that finds restaurants where the first 3 letters of the name are "'Wil": Fields to show: "Id", "name", "borough", and "cuisine".

```
{ _id: ObjectId("60a6d88ff0780227f4caf65b"),
  borough: 'Brooklyn',
  cuisine: 'Delicatessen',
  name: 'Wilken\'S Fine Food',
  restaurant_id: '40356483' }
{ _id: ObjectId("60a6d88ff0780227f4caf65e"),
  borough: 'Bronx',
  cuisine: 'American ',
  name: 'Wild Asia',
  restaurant_id: '40357217' }
{ _id: ObjectId("60a6d891f0780227f4cb0463"),
  borough: 'Bronx',
  cuisine: 'Pizza',
  name: 'Wilbel Pizza',
  restaurant_id: '40871979' }
```


16) Write a query in MongoDB that finds restaurants where the last 3 letters of the name are "'ces": Fields to show: "Id", "name", "borough", and "cuisine".


```
{ _id: ObjectId("60a6d890f0780227f4cafae7"),
  borough: 'Manhattan',
  cuisine: 'American ',
  name: 'Pieces',
  restaurant_id: '40399910' }
{ _id: ObjectId("60a6d890f0780227f4cafba6"),
  borough: 'Queens',
  cuisine: 'American ',
  name: 'S.M.R Restaurant Services',
  restaurant_id: '40403857' }
{ _id: ObjectId("60a6d890f0780227f4cafbac"),
  borough: 'Manhattan',
  cuisine: 'American ',
  name: 'Good Shepherd Services',
  restaurant_id: '40403989' }
{ _id: ObjectId("60a6d891f0780227f4cb005f"),
```


17) Write a query in MongoDB that finds restaurants whose name contains "'Reg" anywhere in the name: Fields to show: "Id", "name", "borough", and "cuisine".

```
db.RESTAURANTS.find
(
    {"name": /.*Reg.*/},
    {
      "restaurant_id" : 1,
      "name":1,"borough":1,
      "cuisine" :1
    }
)
```

18) Write a query in MongoDB that finds restaurants where the "borough" field is "Bronx" or prepares "American" or "Chinese" dishes".

19) Write a query in MongoDB that finds restaurants where the "borough" field is "Staten Island" or "Queens" or "Bronxor" or "Brooklyn". Fields to list: "Id", "name", "borough", and "cuisine"

```
db.RESTAURANTS.find
(
   {"borough" :{$in :["Staten Island","Queens","Bronx","Brooklyn"]}},
   {
     "restaurant_id" : 1,
     "name":1,
     "borough":1,
     "cuisine" :1
   }
)
```

20) Write a query in MongoDB that finds all restaurants, except those located in the boroughs of "Staten Island", "Queens", "Bronx", or "Brooklyn". Fields to list: "Id", "name", "borough", and "cuisine".

21) Write a query in MongoDB that finds all restaurants whose score is not higher than 10. Fields to list: "Id", "name", "borough", "cuisine", and "score".

```
db.RESTAURANTS.find
(
   {"grades.score" : {$not: {$gt : 10}}},
   {"restaurant_id" : 1,"name":1,"borough":1,"cuisine" :1,
"grades.score":1
   }
);
```

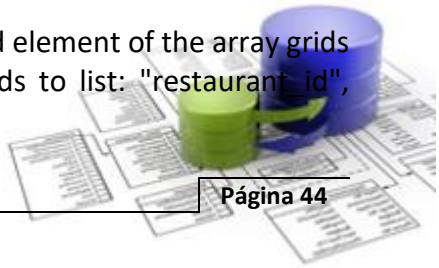
```
{ _id: ObjectId("60a6d88ff0780227f4caf65f"),
  borough: 'Brooklyn',
  cuisine: 'American ',
  grades: [ { score: 5 }, { score: 2 }, { score: 5 }, { score: 2 } ],
  name: 'C & C Catering Service',
  restaurant_id: '40357437' }
{ _id: ObjectId("60a6d88ff0780227f4caf661"),
  borough: 'Manhattan',
  cuisine: 'American ',
  grades: [ { score: 3 }, { score: 4 }, { score: 6 }, { score: 0 } ],
  name: '1 East 66Th Street Kitchen',
  restaurant_id: '40359480' }
{ _id: ObjectId("60a6d88ff0780227f4caf665"),
  borough: 'Brooklyn',
  cuisine: 'Delicatessen',
  grades: [ { score: 4 }, { score: 3 }, { score: 10 } ],
  name: 'Nordic Delicacies'
```

22) Write a query in MongoDB that finds restaurants that do not prepare "American" and "Chinees" dishes or restaurants whose name begins with "Wil". Fields to list: "restaurant_id", "name", "borough", and "cuisine".

23) Write a query in MongoDB that finds the restaurants that have grade "A" and obtained a score of 11 points in ISODate "2014-08-11T00:00:00Z". Fields to list: "restaurant_id", "name", and "grids".

```
db.RESTAURANTS.find(
    {grades:
          {
               $elemMatch:{
                    "date": new ISODate("2014-08-11T00:00:00Z"),
                    "grade" : "A",
                    "score" : 11}
          }
    },
    {"restaurant_id" : 1,"name":1,"grades.$":1}
);
```

24) Write a query in MongoDB that finds the restaurants where: the 2nd element of the array grids is "A", 9 score points, and ISODate "2014-08-11T00:00:00Z". Fields to list: "restaurant_id", "name", and "grids".



```
db.RESTAURANTS.find
(
   { "grades.1.date": ISODate("2014-08-11T00:00:00Z"),
     "grades.1.grade":"A" ,
     "grades.1.score" : 9
   },
   {"restaurant_id" : 1,"name":1,"grades":1}
);
```

```
{ _id: ObjectId("60a6d890f0780227f4cafc7f"),
  grades:
   [ { date: 2015-01-12T00:00:00.000Z, grade: 'A', score: 10 },
     { date: 2014-08-11T00:00:00.000Z, grade: 'A', score: 9 },
     { date: 2014-01-14T00:00:00.000Z, grade: 'A', score: 13 },
     { date: 2013-02-07T00:00:00.000Z, grade: 'A', score: 10 },
     { date: 2012-04-30T00:00:00.000Z, grade: 'A', score: 11 } ],
  name: 'Club Macanudo (Cigar Bar)',
  restaurant_id: '40526406' }
```

25) Write a query in MongoDB that finds restaurants where the 2nd element of the coord array contains a value greater than 42 and less than 52. Fields to list: "restaurant_id", "name", "adress", and "geographical location".

```
db.RESTAURANTS.find
(
   { "address.coord.1": {$gt : 42, $lte : 52}},
   {"restaurant_id" : 1, "name":1, "address":1, "coord":1}
);
```

26) Write a query in MongoDB that shows the restaurants sorted by the name field in ascending order.

27) Write a query in MongoDB that shows the restaurants sorted by the name field in descending order.

28) Write a query in MongoDB that shows restaurants sorted by type of cuisine (ascending) and by neighborhood (borough) in descending order.

29) Write a query in MongoDB that shows all restaurants that contain street (street).

```
db.RESTAURANTS.find
(
   {"address.street" : { $exists : true }}
)
```

30) Write a query in MongoDB that selects the documents from the RESTAURANT collection where the value of the "coord" field is of type DOUBLE

```
db. RESTAURANTS.find
(
   {"address.coord" : {$type : 1}}
)
```

Type	Number	Alias
Double	1	"double"
String	2	"string"
Object	3	"object"
Array	4	"array"
Binary data	5	"binData"

Exercício 8 | Tripadvisor

1) Create the “TRIPADVISOR” database and a collection named “EVALUATION”.

2) Import data from the “EVALUATION.json” file to the “EVALUATION” collection.

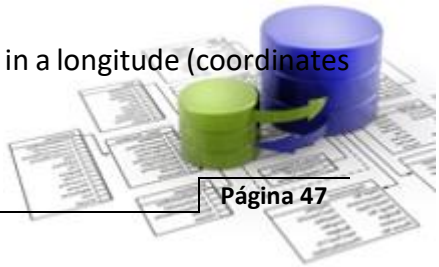
a) Collection structure “EVALUATION”:


```
_id: 83419
contact: Object
    GooglePlaces: null
    Foursquare: "https://foursquare.com/v/pe%C3%B1a-festayre-paris-%C3%AEledefrance/4ad…"
name: "Peña Festayre"
location: Object
    city: "Paris"
    coord: Object
        coordinates: Array
            0: 2.3860357589657
            1: 48.896621743257
        type: "Point"
    address: "80 Boulevard Macdonald"
category: "restaurant"
description: ""
reviews: Array
    0: Object
    1: Object
        wordsCount: 20
        rating: 0
        language: "fr"
        details: "http://tour-pedia.org/api/getReviewDetails?id=52a74a85ae9eef5a50671b09"
        source: "Foursquare"
        text: "Tous les mercredis jusqu'en juin : Soirées Salsa... concert live puis …"
        time: "2012-01-11"
        polarity: 5
    2: Object
nbReviews: 3
```


3) Write a query that shows all documents where the category is "accommodation" and the city is not "Paris" or "London”.

```
db.EVALUATION.find
(
  {$and:
    [
      {"category": 'accommodation'},
      {"location.city":{$nin:['Paris','London']}}
    ]
  }
)
```

4) Write a query that shows all restaurants with a rating <=3 and located in a longitude (coordinates [0]) greater than 2.378938.



```
db.EVALUATION.find
(
  {$and:
    [
      {"reviews.rating": {$lte: 3}},
      {"location.coord.coordinates.0":{$gt: 2.378938}}
    ]
  }
)
```

5) Write a query that allows you to view the first 3 restaurants in the city "Paris”

6) Write a query that allows you to view the 8th and 9th hotels (accommodation) in the city "Paris”.

7) Write a query that allows you to view all documents in the "poi" category where at least 1 of the reviews is greater than 4 (rating). Fields to show: "category", "city", and "coord”.

8) Write a query that allows you to view all restaurants and "poi" (except the city "London") where the total reviews are greater than 5 and less than 10. Fields to show: "name" and "address”.

9) Write a query in MongoDB that finds all the documents in which you have reviewed (time) in January 2010 for the category "restaurant". Fields to show: "name" and "address".

10) Escrever uma query que nos permita visualizar todos os documentos da categoria “poi” e “attraction” em que a “polarity” seja maior que 9 ou o “wordscount” seja menor que 20 (exceto a cidade Paris). Os resultados devem ser ordenados pelo “nbReviews” de forma decrescente.

11) Write a query that shows all documents in the "accommodation" category located in "Rome" (except those with nbReviews less than 5). Results ordered by latitude ("coordinates(1)"), in an ascending fashion. Fields to show: "website" and "city".

12) Write a query that shows all documents sorted by the "nbReviews" field ascending and the "name" field descending. Fields to show: "name", "address", "coord", and "nbReviews".

13) Write a query that finds all documents where the comments (text) include the word "great" or "good". Fields to show: "name" and "text". All categories except "poi".

14) Write a query that allows you to view all hotels (category "accommodation") whose name starts with "Ho", located in "Rome" or "London". Fields to show: "name", "address", and "coord".

15) Write a query that shows the documents in which the reviews were performed in English (language: "en"), or the total words (wordscount) is greater than 15 and the latitude is between 48 and 48.9.

16) Write a query that allows you to identify restaurants located in Paris that do not have any reviews or in the evaluation (text) have the word "bad". The results should be sorted in alphabetical-reverse order of the name ("name").

17) Write a query that allows you to view restaurants that do not have a website or phone and have comments in Portuguese (language:'pt'). Fields to show: "name", "city" and "nbReviews".

18) Write a query that finds all "poi" ("category" field) whose review is between 20 and 60 words ("wordcount") and "polarity" greater than or equal to 6. Fields to show: "name", "wordscount" and "polarity".

19) Write a query that finds the hotels ("accommodation") where the 3rd element of the "reviews" array has the following values: wordscount="10"; polarity="10". Fields to show: "name", "address", and "coord".

20) Write a query that finds restaurants where the 2nd element of the "review" array has the following values: rating = 5 and wordsCount ">20 and <=120".

21) Write a query that shows the "address", "coord", and "nbReviews" fields of all documents in the EVALUATION collection, whose category is "restaurant" and the coordinates are: longitude [0] >=2 and latitude (1) <=49.

22) Write a query that indicates the name of the "poi" with the fewest reviews.

```
db.EVALUATION.find
```

```
(
  {category: 'poi'},
) .sort({"nbReviews":1}).limit(1)
```

OR $group aggregation operators com $min

23) Write a query that selects the document corresponding to the comment with the highest number of words ("wordscount") published between October and November 2013.

```
db.EVALUATION.find
(
 {$and:
   [
    {"category": 'restaurant'},
    {reviews: {$elemMatch: {"time": {$gte: '2013-10-01',$lte: '2013-
11-31'}}}}
   ]
 }
).sort ({"reviews.wordsCount":-1}).limit(1)
```

Ou $group aggregation operators com $max

24) Write a query that calculates the total number of restaurants in Rome that have more than 20 reviews in the year 2012.

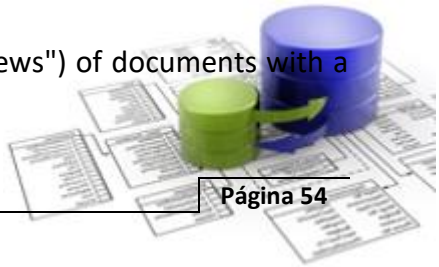
25) Write a query that shows the top 5 restaurants whose "nbReviews" is even. Fields to show: name, city, and address of the restaurant. Fields to show: "name", "nbReviews", and "coord".

26) Write a query that changes the category of the restaurant "Starbucks" whose address is "21 Rue des Petits Carreaux, Paris, France" to "cafeteria", and the coordinates for 2.347244 and 48.867244.

27) Write a query that allows you to add a new field ("ano_analise": "2022") to all documents in the collection. Also, write a new query that allows you to add a new field ("website ":"www.xpto.pt"), to the restaurant with ID 84820

28) Write a query that removes 3 values in the review rating 16 (it has to go from 5 to 2) of the restaurant with the name "Boulangerie Pichard" located in Paris.

29) Write a query that shows the average number of reviews ("nbReviews") of documents with a rating of less than 3 – By city.



```
{ _id: 'Paris', media_nbreviews: 5.2901876884177765 }
{ _id: null, media_nbreviews: 10.521428571428572 }
{ _id: 'Rome', media_nbreviews: 19.5 }
```

```
db.EVALUATION.aggregate
(
  [
    {$match: {'reviews.rating': {$gt: 3}}},
    {$group:
      {
        _id: "$location.city",
        media_nbreviews: {$avg: "$nbReviews"}
      }
    }
  ]
)
```

30) Write a query that shows the average rating of the restaurant "Boulangerie Pichard" (_id: 84820).

Exercício 9 | Sales

1) Create the database “vendetudo" and a collection with the name " VENDAS ".

a) Create the Database “vendetudo”.

b) Create the collection “VENDAS”.

2) Import data from the file “VENDAS.json” to the collection “VENDAS”.

Collection structure “VENDAS”:


```
_id: ObjectId("5bd761dcae323e45a93ccfe8")
saleDate: 2015-03-23T21:06:49.506+00:00
items: Array
  0: Object
    name: "printer paper"
    tags: Array
      0: "office"
      1: "stationary"
    price: 40.01
    quantity: 2
  1: Object
    name: "notepad"
    tags: Array
    price: 35.29
    quantity: 2
  2: Object
  3: Object
  4: Object
  5: Object
  6: Object
  7: Object
storeLocation: "Denver"
customer: Object
  gender: "M"
  age: 42
  email: "cauho@witwuta.sv"
  satisfaction: 4
couponUsed: true
purchaseMethod: "Online"
```


3) Write a query in MongoDB that shows all the documents in the collection VENDAS.

4) Write a query in MongoDB that shows the "saleDate", "storeLocation", and "purchaseMethod" fields of all documents in the collection VENDAS.

5) Write a query in MongoDB that shows the fields "saleDate", "storeLocation", and "purchaseMethod" but excludes the "_id" field from all documents in the collection VENDAS

6) Write a query in MongoDB that shows the "saleDate", "storeLocation" and "gender" fields, but excludes the "_id" field from all documents in the collection VENDAS

or

7) Write a query in MongoDB that shows all SALES where the "storeLocation" field is "Denver".

8) Write a query in MongoDB that shows the first 2 SALES in the store ("storeLocation") of "Denver".

9) Write a query in MongoDB that skips the first 2 SALES and shows the next 4 where the "storeLocation" field is "Denver".

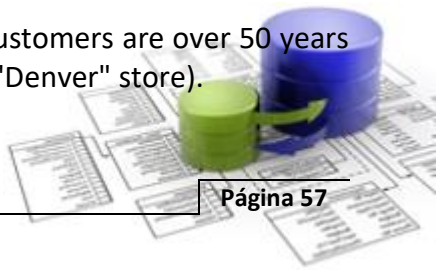
10) Write a query in MongoDB that finds SALES where at least 1 of the items has sold more than 9 units ("quantity").

or

11) Write a query in MongoDB that finds sales where the quantity purchased is greater than 6 and less than 8. Fields to show: "saleDate" and "storeLocation".

12) Write a query in MongoDB that finds sales made by customers under the age of 18.

13) Write a query in MongoDB that allows us to visualize sales where customers are over 50 years old and the purchase method is "Phone" (except for sales from the "Denver" store).

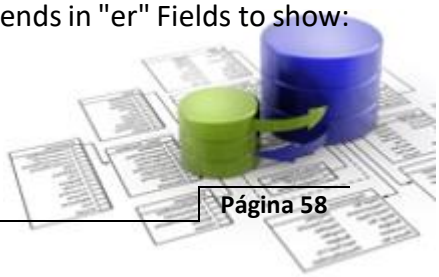


a) without using the $and operator

14) Write a query in MongoDB that shows all printer paper sales made in "London" (except those made by "Phone"). Ordered by date ("saleDate"), in descending order.

15) Write a query in MongoDB that finds all sales whose "storeLocation" starts at "Lo" Fields to show: "storeLocation" and "gender".

16) Write a query in MongoDB that finds all sales whose "storeLocation" ends in "er" Fields to show: "storeLocation" and "gender".



17) Write a query in MongoDB that finds all sales where the name of the "storeLocation" contains "le" anywhere. Fields to show: "storeLocation" and "gender".

18) Write a query in MongoDB that shows sales made in "Seattle" or "London" where the customer satisfaction level was greater than 4.

19) Write a query in MongoDB that finds all sales made in "San Diego", "New York" and "Austin". Fields to list: "saleDate", "email", and "purchaseMethod". The results should be sorted alphabetically by location ("storeLocation").

20) Write a query in MongoDB that finds all sales, except those made in "San Diego", "New York", and "Austin". Fields to list: "saleDate", "email", and "purchaseMethod". The results should be sorted in alphabetical-reverse order of location ("storeLocation").

21) Write a query in MongoDB that finds all sales in which the degree of satisfaction is not less than 3 and the age between 20 and 40 years. Fields to list: "saleDate", "storeLocation", "satisfaction" and "age".

22) Write a query in MongoDB that finds all sales from customers whose email does not end in "pt" and the purchaseMethod is not "Phone" or "Online". Fields to list: "purchaseMethod", "storeLocation", "satisfaction", and "email".

23) Write a query in MongoDB that finds all sales in which envelopes were purchased in an amount equal to 6. Fields to show: "storeLocation" and "gender".

24) Write a query in MongoDB that finds sales where the 1st element of the "items" array has the following values: name="laptop"; quantity="2". Fields to show: "storeLocation" and "gender".

25) Write a query in MongoDB that finds sales where the 2nd element of the "items" array has the following values: name="laptop"; price=">1200 and <1600". Fields to show: "storeLocation" and "age".

26) Write a query in MongoDB that shows sales sorted by the "satisfaction" field in ascending order.

27) Write a query in MongoDB that shows sales sorted by the "storeLocation" field in descending order.

28) Write a query in MongoDB that shows sales sorted by the "storeLocation" field (ascending) and the "satisfaction" field in a descending manner.

29) Write a query in MongoDB that shows all sales where only 1 item was sold.

30) Write a query in MongoDB that selects the document corresponding to the oldest client.

31) Write a query in MongoDB that shows the total number of customers by gender.

```
db.VENDAS.aggregate
(
  [
  {
    $group: {
```

```
      _id: "$gender",
      totalSales: { $sum: 1 }
    }
  }
])
```

`$sum operator is used to calculate the total sales by incrementing the count by 1 for each document in the group.`

32) Write a query in MongoDB that shows the total sales by city.

> NOTE: ver $unwind e $multiply

```
db.VENDAS.aggregate([
  {
    $unwind: "$items"
  },
  {
    $group: {
      _id: "$city",
      totalSales: { $sum: { $multiply: [ "$items.quantity",
"$items.price" ] } }
    }
  }
])
```

> $unwind stage to deconstruct the amounts array.
> Then, in the $group stage, group the documents by the city field. The _id field represents the grouping key, which is the city field. To calculate the total sales, we use the $multiply operator to multiply the quantity and unit_price fields for each document in the group. The $sum operator then calculates the sum of the products. Executing this query will return a result that includes the _id field representing the city and the totalSales field representing the total sales amount for each city, taking into account both the quantity and unit_price fields.

Exercício 10 | Football Championship

1) Create the database "LIGAS" and a collection with the name "EQUIPAS".

a) Create the Database "LIGAS".

b) Create the collection “EQUIPAS”.

2) Import data from the file “equipas.json” to the collection "EQUIPAS".

Collection structure "EQUIPAS":


```
_id: ObjectId('647db37f3fd0fc0a7a74c029')
equipa_id: "1"
nome: "Sporting Clube de Portugal"
fundacao: "1906"
▾ morada: Object
    cidade: "Lisboa"
    rua: "Alvalade n◆mero 1"
    codigopostal: "1000-001"
liga: "Liga NOS"
▾ jornadas: Array
  ▾ 0: Object
      data: 2020-10-28T00:00:00.000+00:00
      jornada: 1
      resultado: "V"
      GM: 3
      GS: 1
      PT: 3
  ▸ 1: Object
  ▸ 2: Object
  ▸ 3: Object
  ▸ 4: Object
  ▸ 5: Object
  ▸ 6: Object
  ▸ 7: Object
  ▸ 8: Object
  ▸ 9: Object
  ▸ 10: Object
```


3) Write a query in MongoDB that shows all the documents in the collection EQUIPAS.

4) Write a query in MongoDB that shows the fields "equipa_id", "nome", "liga", and "fundacao" of all documents in the collection EQUIPAS.

5) Write a query in MongoDB that shows the fields "equipa_id", "nome", "liga", and "fundacao", but hides the "_id", of all documents in the collection EQUIPAS

6) Write a query in MongoDB that shows the fields "equipa_id", "nome", "liga", and "codigopostal", but hides the "_id", of all documents in the collection EQUIPAS

7) Write a query in MongoDB that shows all the teams in which the field "liga" is "Liga NOS".

8) Write a query in MongoDB that shows the first 5 teams where the field "liga" is "Liga NOS".

9) Write a query in MongoDB that skips the first 5 teams and shows the next 5 in that field "liga" is "Liga NOS".

10) Write a query in MongoDB that finds the teams that have scored more than 78 points "PT" in the respective championship.

11) Write a query in MongoDB that finds the teams that have scored more than 20 points "PT" and less than 33 points in “Jornada” number 34. Fields to display: "nome", "cidade" and "liga".

12) Write a query in MongoDB that finds the teams that are located in the city of "Lisboa". Fields to display: "nome", "cidade", "liga" and "fundacao".

13) Write a query in MongoDB that finds the teams that are not from the league "Liga NOS", where the score "PT" is greater than 70 on "jornada" number 33. Fields to display: "nome", "cidade", "liga", "PT", and only the data from “jornada” number 33.

a) Write the query without using the $and

14) Write a query in MongoDB that finds the teams for which “liga” is not "Liga NOS", the score "PT" is greater than 50 in "jornada" number 32, and do not reside in Porto ("cidade"). The fields to show: "nome", "cidade", "fundacao", "liga", "PT", with the data only from “jornada” number 32, and must be presented in descending order of the year of the team's foundation "fundacao".

15) Write a query in MongoDB that finds teams where the first 4 letters of the name are "Spor". The fields to show: "equipa_id", "nome", "cidade" e "liga".

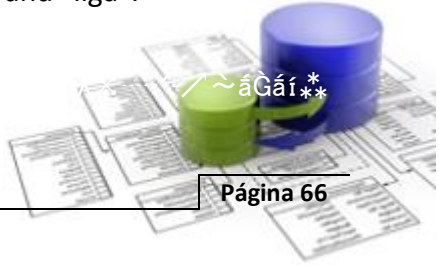
16) Write a query in MongoDB that finds teams where the last 3 letters of the name are "nse": The fields to show: "equipa_id", "nome", "cidade" e "liga".

17) Write a query in MongoDB that finds the teams whose name contains "ort" anywhere in the name. The fields to show: "equipa_id", "nome", "cidade" e "liga".

18) Write a query in MongoDB that finds the teams in which the field "cidade" is "Porto" or "Lisboa", and that the league (Liga) is "Liga NOS". The fields to show: "nome", "cidade" e "liga".

19) Write a query in MongoDB that finds the teams in which the field "cidade" is "Chaves" or "Faro", "Coimbra" or "Funchal". The fields to show: "nome", "cidade" and "liga".

20) Write a query in MongoDB that finds all teams except those located in the cities of "Porto", "Faro", "Lisboa", or "Funchal". The fields to show: "nome", "cidade" and "liga".



21) Write a query in MongoDB that finds all teams whose score "PT" was not more than 30. The fields to show: "nome", "cidade" e "liga".

22) Write a query in MongoDB that finds teams whose name starts with "Des" or whose city is not "Porto" or "Lisboa". Fields to list: "equipa_id", "nome", "cidade" and "liga".

23) Write a query in MongoDB that finds the teams that have marked "GM" two (2) goals but a loss at ISODate "2021-02-13T00:00:00Z". Fields to lists: "equipa_id", "nome", "liga" and respective "jornada".

24) Write a query in MongoDB that finds the teams where: The 10th element of the array "jornadas" has as result ("resultado") a victory "V" and ISODate "2020-12-19T00:00:00Z". Fields to list: "equipa_id", "nome", "data", "jornada" and "resultado" from array "jornadas".

25) Write a query in MongoDB that finds the teams in which the 4th element of the array "jornadas" contains a point value (PT) greater than 8 and less than 10. Fields to list: "equipa_id", "nome" and "PT".

26) Write a query in MongoDB that shows the teams sorted by the name field (nome) in ascending order. Fields to list: "nome".

27) Write a query in MongoDB that shows the teams sorted by the name field in descending order.

28) Write a query in MongoDB that shows the teams sorted by name (ascendant) and year of foundation (fundação) descending. Fields to list: "nome" and "fundacao".

29) Write a query in MongoDB that shows all teams that contain the field "rua". Fields to list: "nome" and not “jornadas”.

30) Write a query in MongoDB that selects the documents in the collection EQUIPAS where the value of the field "resultado" is of the type STRING. Fields to list: "nome" and not “jornadas”.

31) Write a query in MongoDB that shows the total goals scored (field “GM”) by the team. Sort in descending order.

use $unwind

```
db.equipas.aggregate
(
  [
```

```
    {$unwind: "$jornadas"},
    {
     $group:
     {
       _id: "$nome",
       total_golos: { $sum: "$jornadas.GM" }
     }
    }
   ]
)
```

32) Write a query in MongoDB that shows the team with the highest difference between goals scored and goals conceded (field “GM” and “GS”) at the end of all matchdays (at the end of the championship). Sort in descending order.

$unwind and $subtract

Exercício 11 | Student grades

1) Create a database and insert 4 documents with the following structure:

Example of 1 document from "evaluation" collection:


```
_id: ObjectId("50b59cd75bed76f46522c34e")
student_id: 1234
scores: Array
  0: Object
      type: "exam"
      score: 57.92947112575566
  1: Object
      type: "quiz"
      score: 21.24542588206755
  2: Object
      type: "homework"
      score: 68.1956781058743
  3: Object
      type: "homework"
      score: 67.95019716560351
  4: Object
      type: "homework"
      score: 18.81037253352722
class: Object
    class_id: 2
    class_name: "Databases"
```


2) Write a query in MongoDB that indicates the total number of courses attended by the student with ID=1234.

```
db.evaluation.count({student_id:1234})

or

db. evaluation.find({student_id:1234}).count()
```

3) Write a query in MongoDB that shows all students who obtained a "score" greater than 50 in the "exam" of the subjects ("class_name") named "Databases" and "Programming", or a "score" greater than 40 in the "homework" of the disciplines ("class_name") named "Oracle", "SQL" and "Programming",. Fields to show: "student_id" and ""class_name".

4) Write a query in MongoDB that shows all documents where the discipline name ("class_name") ends in "ses" (except for "Databases") and has a score lower than 70 in the exam or work. Fields to show: "student_id", "type" and "class_name".

5) Write a query in MongoDB that indicates the ID of the student with the worst grade in the exam of "Databases".

6) Write a query in MongoDB that updates, in the class_name “Databases”, the “score” of the students' exam with ID1234 and ID4321 to 75.55.

Exercício 12 | Products

1) Create DB “produtos” and collection “encomendas”

2) Insert 5 documents:

The following collection of five documents is given. Documents consist of orders. An order has an id (e.g. “o1”), the year in which it was issues, the cost, the items in the order, and the number of days it took to deliver the order. The cost is specified as price in a given currency. The order items consist of products. A product has an id (e.g., “p1”), colours, and quantity.

```
{"order":"o1","year":2020,"paid":"Y","cost":{"price":30,"currency":"NO
K"}, "items":[{"product":"p1","colours":["blue","black"] ,
"quantity":15}] , "delivery_days":5}

{"order":"o2","year":2020,"paid":"Y","cost":{"price":13,"currency":"EU
R"}, "items":[{"product":"p2","colours":["white"],"quantity":4},
{"product":"p3","colours":["white","black"],"quantity":1}],"delivery_d
ays":4},

{"order":"o3","year":2018,"paid":"N","cost":{"price":33,"currency":"EU
R"},items":[{"product":"p3","colours":["blue","black"],"quantity":4}],
"delivery_days":4},

{"order":"o4","year":2017,"paid":"Y","cost":{"price":17,"currency":"NO
K"},"items":[{"product":"p2","colours":["pink","black"],"quantity":14}
,{"product":"p4","colours":["white"],"quantity":1}],
"delivery_days":2},

{"order":"o5","year":2020,"paid":"Y","cost":{"price":19,"currency":"NO
K"},"items":[{"product":"p1","quantity":15}], "delivery_days":3}
```

3) Remove document with “order”: “o3”.

```
db.encomendas.deleteOne({ "order": "o3" })
```

4) Insert a new document.

```
{
  "order": "o6",
  "year": 2020,
  "paid": "N",
  "cost": {
    "price": 40,
    "currency": "EUR"
  },
  "items": [
    {
      "product": "p2",
      "colours": ["pink", "black"],
      "quantity": 14
```

```
      },
      {
        "product": "p4",
        "colours": ["white"],
        "quantity": 1
      }
    ],
    "delivery_days": 5
  }
```

5) Show all documents in a collection

6) Show all documents that contain paid orders (the "paid" field is "Y")

7) Show all documents that contain paid orders, and the orders are from before 2019.

8) Show all documents that contain unpaid orders or whose orders are from before 2019.

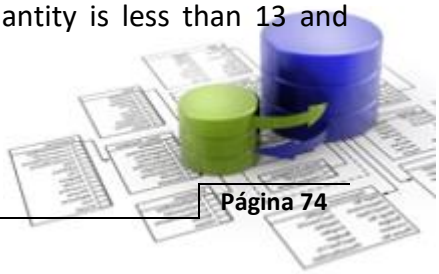
9) Show all documents that contain orders whose price is in NOK

10) Show all documents that contain orders whose price is less than 18 NOK

11) Show all documents with orders that contain product "p2"

12) Show all documents with orders that contain products whose quantity is less than 13

13) Show all documents with orders that contain products whose quantity is less than 13 and contain no products whose quantity exceeds 13.



14) Showall documents with orders that contain products whose first colour (i.e., first element in the "colours" array) is blue

```
db.encomendas.find({
  "items": {
    $elemMatch: {
      "colours.0": "blue"
    }
  }
});
```

> This query uses the $elemMatch operator to match documents where the "items" array contains at least one element with the first color in the "colours" array equal to "blue". It will return all documents that satisfy this condition.

15) Show the total number of delivery days, grouped by year; retrieve the results only after 2017 (Hint: use aggregation pipelines)

Exercício 13 | Pizzaria

1) Create DB “pizzaria”.

2) Insert three documents into a collection called "users" with the following fields: "name", "age", and "email".

```
db.users.insertMany([
  { name: "John", age: 25, email: "john@example.com" },
  { name: "Jane", age: 30, email: "jane@example.com" },
  { name: "Mike", age: 35, email: "mike@example.com" }
]);
```

3) Update the age of the user with the name "John" to 26.

4) Delete the user with the email "jane@example.com".

5) Find the average age of all users in the collection.

6) Find all users whose age is greater than 30.