MCA Department

NoSQL Databases Lab (18MCAR306L) Record

Submitted by

MANGALORE KAUSHAL KISHORE RAO (18MCAR0034)

III Semester, MCA General

Submitted to

MR. PRAVEEN K. PANDEY

JAIN (DEEMED-TO-BE UNIVERSITY)

School of Computer Science and Information Technology

Jayanagar, Bangalore.

Table of Experiments

Experiment 01....................4
Aim....................4
Procedure....................4
Output....................5

Experiment 02....................6
Aim....................6
Procedure....................6
Output....................7

Experiment 03....................9
Aim....................9
Procedure....................9
Output....................10

Experiment 04....................11
Aim....................11
Procedure....................11
Output....................13

Experiment 05....................14
Aim....................14
Procedure....................14
Output....................15

Experiment 06 & 07....................16
Aim (06)....................16
Aim (07)....................16
Procedure....................17
Output....................18

Experiment 08....................20
Aim....................20
Procedure....................20
Output....................21

Experiment 09 22
Aim 22
Procedure 22
Output 22
Experiment 10 23
Aim 23
Procedure 23
Output 24
Experiment 11 25
Aim 25
Procedure 25
Craigslist 25
Cisco 26
Electronic Arts (EA) 27
Thermo Fisher 28
GAP 29

EXPERIMENT 01

Aim

Prepare and install infrastructure for setting up MongoDB lab:

- Install MongoDB Community Edition
  - Download MongoDB Community Edition
  - Run the MongoDB installer
  - Follow the MongoDB Community Edition installation wizard
- Run MongoDB Community Edition as a Windows Service
- Run MongoDB Community Edition from the Command Interpreter

It is advised to follow the URL: https://docs.mongodb.com/manual/tutorial/install-mongodb-on-windows/

Procedure

As per the URL, we get these instructions to follow:

1. **Install MongoDB Community Edition:**
   a. Download the latest edition of **MongoDB Community Server** (currently version 4.0.x) for the OS (Windows 64-bit x64) as an installer (MSI).
   b. Run the MongoDB installer.
   c. Follow the MongoDB Community Edition installation wizard, making these choices:
      i. Select **'Complete'** installation.
      ii. Select **'Install MongoD as a Service'** option and choose **'Run service as a Network Service user'**, with the default settings.
      iii. (Optional) Install MongoDB Compass.

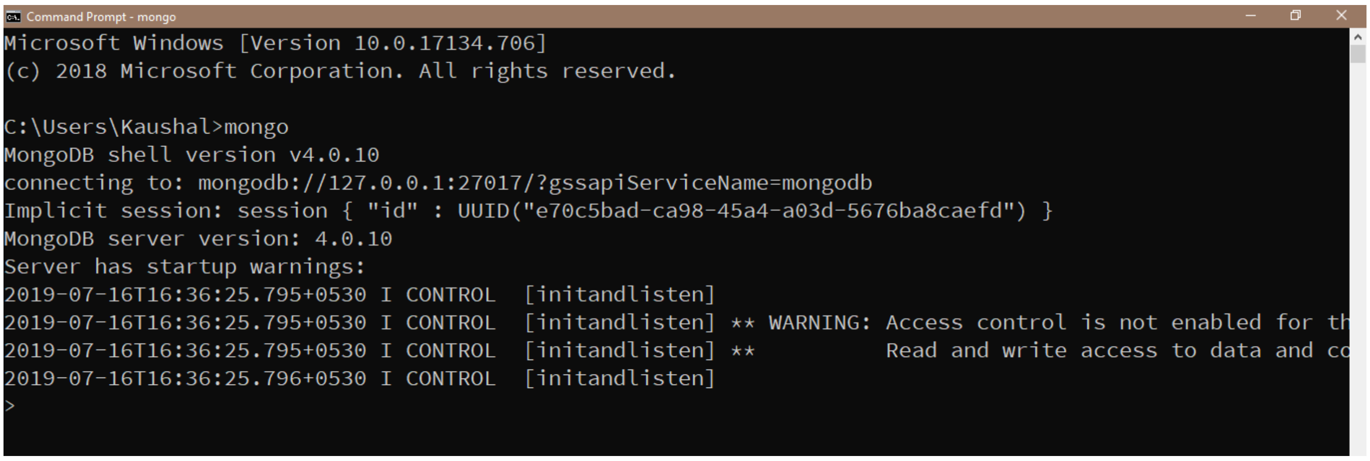
2. **Run MongoDB Community Edition as a Windows Service:**
   a. Open the Services App, and find the **'MongoDB Server'** service.
   b. Right-click the service and select the option to **'Start'** the service.

3. **Run MongoDB Community Edition from the Command Interpreter:**
   a. Create the database directory at **'C:\data\db'**.

b. Open the Command Interpreter with Administrative privileges and add the path of the database directory to MongoDB using the command: **'"C:\Program Files\MongoDB\Server\4.0\bin\mongod.exe" --dbpath="c:\data\db"'**.
c. Then run **'"C:\Program Files\MongoDB\Server\4.0\bin\mongo.exe"'** to start MongoDB.
d. (Optional) Add the above command to the System Path.

Output

The output of this experiment is the successful installation of MongoDB on the system. This means that MongoDB can be run from the Command Interpreter as shown below:



```
Microsoft Windows [Version 10.0.17134.706]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Kaushal>mongo
MongoDB shell version v4.0.10
connecting to: mongodb://127.0.0.1:27017/?gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("e70c5bad-ca98-45a4-a03d-5676ba8caefd") }
MongoDB server version: 4.0.10
Server has startup warnings:
2019-07-16T16:36:25.795+0530 I CONTROL  [initandlisten]
2019-07-16T16:36:25.795+0530 I CONTROL  [initandlisten] ** WARNING: Access control is not enabled for th
2019-07-16T16:36:25.795+0530 I CONTROL  [initandlisten] **          Read and write access to data and co
2019-07-16T16:36:25.796+0530 I CONTROL  [initandlisten]
>
```

EXPERIMENT 02

Aim

Perform or execute the following basic commands in the MongoDB lab environment:

- Login to Lab
- Show all Databases
- Select database to work with
- Authenticate and Log out from databases
- List down Collections, Users, Roles
- Create Collection

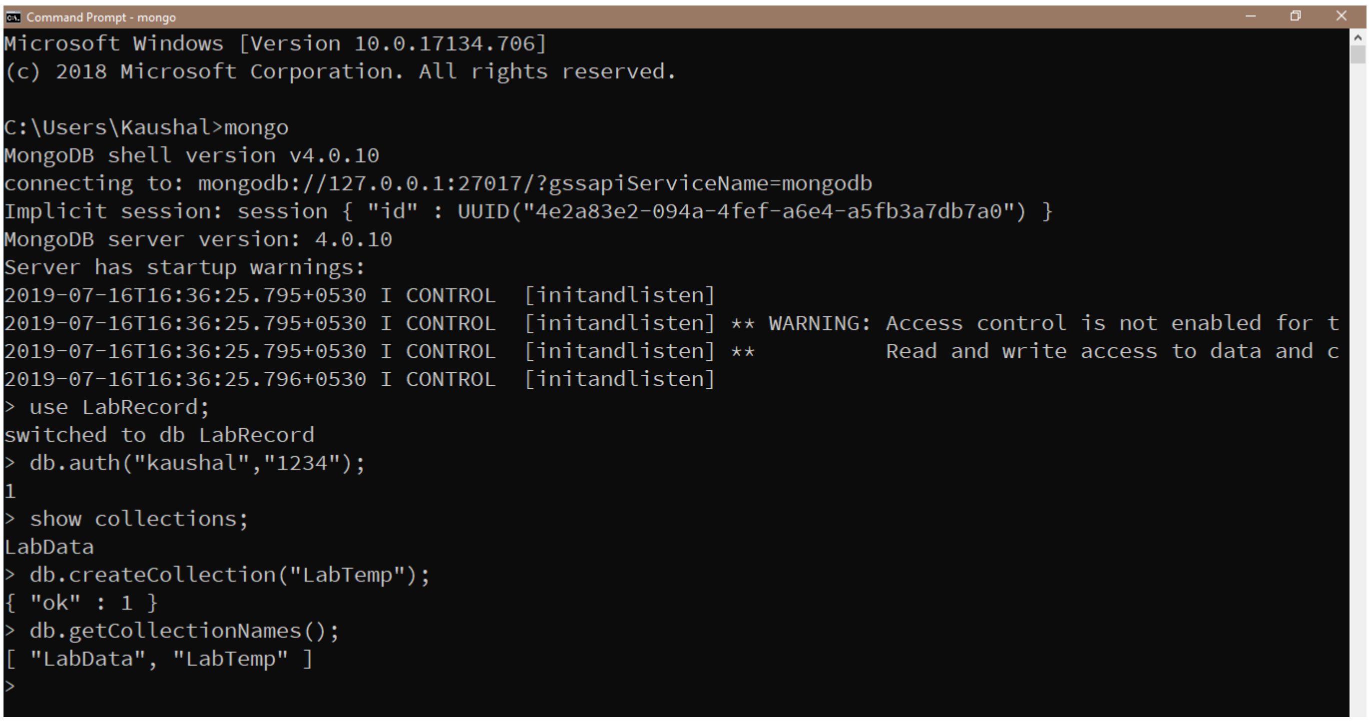
Procedure

Here's how to perform each of the specified commands in MongoDB:

1. Login to Lab:
    a. We first create a database called Lab with the command **'use LabRecord;'**.
    b. Then create a user with which to login, **'db.createUser({ name: "kaushal", pwd: "1234", roles: ["readWrite", "dbAdmin"] });'**.
    c. Exit from MongoDB and start it again.
    d. We go to Lab with the command **'use LabRecord;'**, and then login with **'db.auth("kaushal", "1234");'**.
2. Show all databases with the command **'show dbs;'**.
3. Select database to work with using **'use <databaseName>;'**, for example **'use LabRecord;'**.
4. Authenticate database by typing **'db.auth ("kaushal", "1234");'** and logout with **'db.logout( );'**.
5. To list collection, users and roles we use the following after selecting our database:
    a. List collection – **'show collections;'** or **'db.getCollectionNames( );'**.
    b. List users – **db.getUsers( );'**.
    c. List roles – **'db.getRoles({ rolesInfo: 1, showPrivileges: true, showBuiltinRoles: true });'**.
6. Create collection by simply typing **'db.createCollection(<collection-name>);'**, for example **'db.createCollection("LabData");'**.

Output

As the output for all the above would be extensive, the below screenshots show a few of the outputs for the above commands:


Command Prompt - mongo

```
Microsoft Windows [Version 10.0.17134.706]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Kaushal>mongo
MongoDB shell version v4.0.10
connecting to: mongodb://127.0.0.1:27017/?gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("4e2a83e2-094a-4fef-a6e4-a5fb3a7db7a0") }
MongoDB server version: 4.0.10
Server has startup warnings:
2019-07-16T16:36:25.795+0530 I CONTROL  [initandlisten]
2019-07-16T16:36:25.795+0530 I CONTROL  [initandlisten] ** WARNING: Access control is not enabled for t
2019-07-16T16:36:25.795+0530 I CONTROL  [initandlisten] **          Read and write access to data and c
2019-07-16T16:36:25.796+0530 I CONTROL  [initandlisten]
> use LabRecord;
switched to db LabRecord
> db.auth("kaushal","1234");
1
> show collections;
LabData
> db.createCollection("LabTemp");
{ "ok" : 1 }
> db.getCollectionNames();
[ "LabData", "LabTemp" ]
>
```

EXPERIMENT 03

Aim

Perform or execute the following basic commands in the MongoDB lab environment:

- Insert Document
- Save Document
- Update Document
- Display Collection Records
- Drop Function

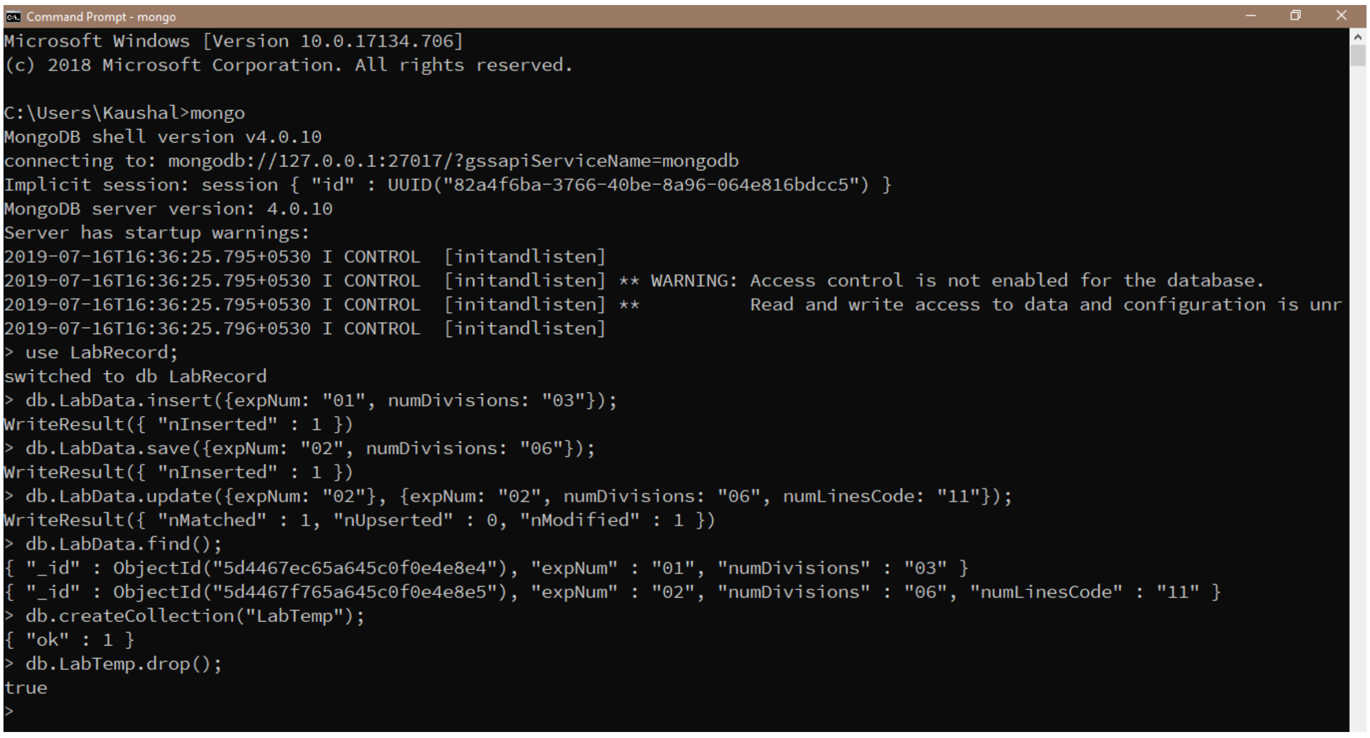
Procedure

Here's how to perform each of the specified commands in MongoDB:

1. Insert one or more documents into a collection using **'db.LabData.insert({ expNum: "01", numDivisions: "03" });'**.
2. Save documents into a collection using **'db.LabData.save({ expNum: "02", numDivisions: "06" });'**.
3. Update documents in a collection using **'db.LabData.update({ expNum: "02" }, { expNum: "02", numDivisions: "06", numLinesCode: "11" });'**.
4. Update documents in a collection using **'db.LabData.update({ expNum: "02" }, { expNum: "02", numDivisions:"06", numLinesCode:"11" });'**.
5. Display records in a collection using **'db.LabData.find( )'**.
6. Drop a collection using **'db.LabTemp.drop( );'**.

Output

The output for the commands would be as follows:


Command Prompt - mongo

```
Microsoft Windows [Version 10.0.17134.706]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Kaushal>mongo
MongoDB shell version v4.0.10
connecting to: mongodb://127.0.0.1:27017/?gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("82a4f6ba-3766-40be-8a96-064e816bdcc5") }
MongoDB server version: 4.0.10
Server has startup warnings:
2019-07-16T16:36:25.795+0530 I CONTROL  [initandlisten]
2019-07-16T16:36:25.795+0530 I CONTROL  [initandlisten] ** WARNING: Access control is not enabled for the database.
2019-07-16T16:36:25.795+0530 I CONTROL  [initandlisten] **          Read and write access to data and configuration is unr
2019-07-16T16:36:25.796+0530 I CONTROL  [initandlisten]
> use LabRecord;
switched to db LabRecord
> db.LabData.insert({expNum: "01", numDivisions: "03"});
WriteResult({ "nInserted" : 1 })
> db.LabData.save({expNum: "02", numDivisions: "06"});
WriteResult({ "nInserted" : 1 })
> db.LabData.update({expNum: "02"}, {expNum: "02", numDivisions: "06", numLinesCode: "11"});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.LabData.find();
{ "_id" : ObjectId("5d4467ec65a645c0f0e4e8e4"), "expNum" : "01", "numDivisions" : "03" }
{ "_id" : ObjectId("5d4467f765a645c0f0e4e8e5"), "expNum" : "02", "numDivisions" : "06", "numLinesCode" : "11" }
> db.createCollection("LabTemp");
{ "ok" : 1 }
> db.LabTemp.drop();
true
>
```

EXPERIMENT 04

Aim

Perform or execute the following advanced commands in the MongoDB lab environment:

- Administrative Commands
- Projection
- Limit Method
- Skip Method
- Sort Records
- Indexing
- Aggregation
- Interacting with cursors

Procedure

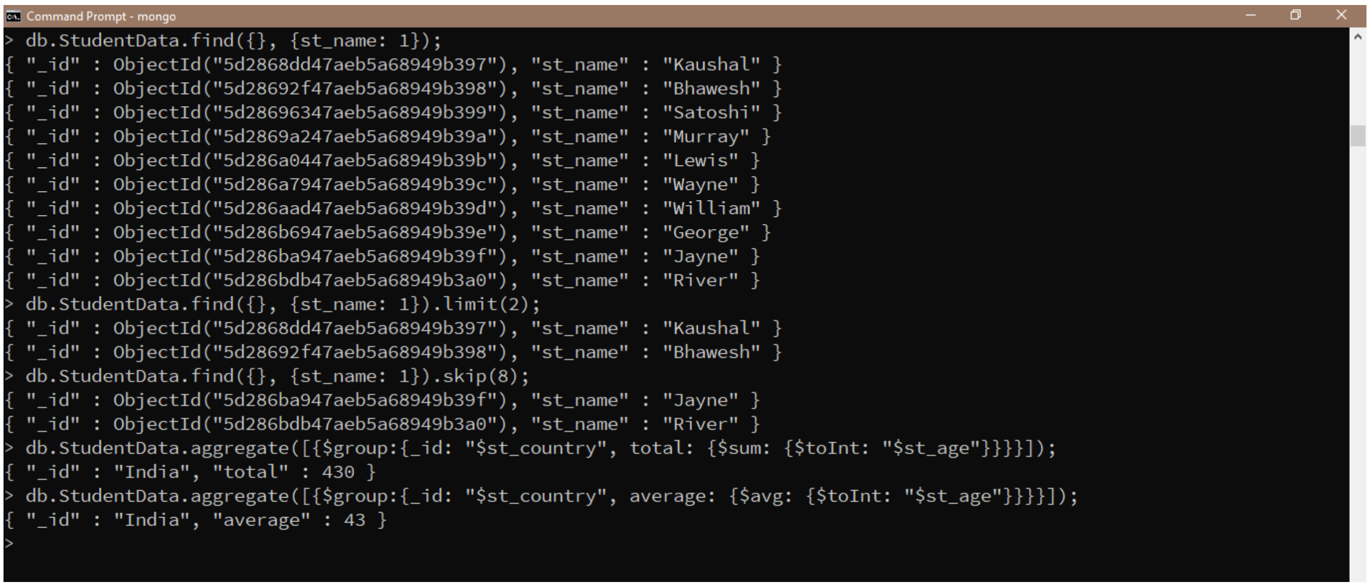
Here's how to perform each of the specified commands in MongoDB:

1. Administrative commands include several commands that perform organisational operations on the database, some of them are:
    a. List all existing databases by using **'db.adminCommand({ listDatabases: 1 });'**.
    b. By using the **'db.adminCommand({ listCollections: 1, nameOnly: true });'** command, all the collections in the current database can be listed.
    c. When a collection needs to be renamed, the **'db.adminCommand({ renameCollection: "LabRecord.LabTemp", to: "LabRecord.TempLab" });'** command can be used.
    d. Drop a collection from the database using **'db.TempLab.drop( );'**.
    e. To drop an entire database and its data, select the database with **'use tempDB;'**, and while in the database run the **'db.dropDatabase( );'** command.
2. Projection is the viewing of the data stored in a collection - **'db.StudentData.find( );'** can be used to get all the data, and **'db.StudentData.find({ }, { st_name: 1 });'** to project a specific field.

3. The Limit method is used to specify the number of results to be projected when displaying the results of a query; this can be done using **‘db.StudentData.find( ).limit(3);’**, which will return the first three results for the query.
4. The Skip method is used to skip a number of results when displaying the results of a query; this can be done using **‘db.StudentData.find( ).skip(5);’**, which will return all the results for the query except the first five.
5. To sort our data according to a field:
   a. We would use **‘db.StudentData.find( ).sort({ st_id: 1 });’** to sort in ascending order.
   b. We would use **‘db.StudentData.find( ).sort({ st_id: -1 });’** to sort in descending order.
6. To set up indexing for the database, we must use **‘db.StudentData.createIndex({st_name: 1});’**.
7. While aggregation function can be used for many insights into the data, we’ll just use a couple of them for simple operations:
   a. We would use **‘db.StudentData.aggregate([{ $group: { _id: "$st_country", total_age: { $sum: "$st_age" } } }]);’** to get the sum of a numerical field by grouping according to the specified field.
   b. We would use **‘db.StudentData.aggregate([{ $group: { _id: "$st_country", average_age: { $avg: "$st_age" } } }]);’** to get the average of a numerical field by grouping according to the specified field.
8. The cursor refers to the results of the find operation which are returned as an iterable list; we can perform many operations on cursors, some of which are:
   a. Count number of results using **‘db.StudentData.find( ).count( );’**.
   b. Limit number of results using **‘db.StudentData.find( ).limit(2);’**.
   c. Adds line breaks to improve readability of results with **‘db.StudentData.find( ).pretty( );’**.
   d. Skip a number of results using **‘db.StudentData.find( ).skip(3);’**.

Output

The above program is extensive, so the output below will only show some of the results:



```
> db.StudentData.find({}, {st_name: 1});
{ "_id" : ObjectId("5d2868dd47aeb5a68949b397"), "st_name" : "Kaushal" }
{ "_id" : ObjectId("5d28692f47aeb5a68949b398"), "st_name" : "Bhawesh" }
{ "_id" : ObjectId("5d28696347aeb5a68949b399"), "st_name" : "Satoshi" }
{ "_id" : ObjectId("5d2869a247aeb5a68949b39a"), "st_name" : "Murray" }
{ "_id" : ObjectId("5d286a0447aeb5a68949b39b"), "st_name" : "Lewis" }
{ "_id" : ObjectId("5d286a7947aeb5a68949b39c"), "st_name" : "Wayne" }
{ "_id" : ObjectId("5d286aad47aeb5a68949b39d"), "st_name" : "William" }
{ "_id" : ObjectId("5d286b6947aeb5a68949b39e"), "st_name" : "George" }
{ "_id" : ObjectId("5d286ba947aeb5a68949b39f"), "st_name" : "Jayne" }
{ "_id" : ObjectId("5d286bdb47aeb5a68949b3a0"), "st_name" : "River" }
> db.StudentData.find({}, {st_name: 1}).limit(2);
{ "_id" : ObjectId("5d2868dd47aeb5a68949b397"), "st_name" : "Kaushal" }
{ "_id" : ObjectId("5d28692f47aeb5a68949b398"), "st_name" : "Bhawesh" }
> db.StudentData.find({}, {st_name: 1}).skip(8);
{ "_id" : ObjectId("5d286ba947aeb5a68949b39f"), "st_name" : "Jayne" }
{ "_id" : ObjectId("5d286bdb47aeb5a68949b3a0"), "st_name" : "River" }
> db.StudentData.aggregate([{$group:{_id: "$st_country", total: {$sum: {$toInt: "$st_age"}}}}]);
{ "_id" : "India", "total" : 430 }
> db.StudentData.aggregate([{$group:{_id: "$st_country", average: {$avg: {$toInt: "$st_age"}}}}]);
{ "_id" : "India", "average" : 43 }
>
```

EXPERIMENT 05

Aim

Execute below steps by inserting some data which we can work with.

Paste the following into your terminal to create a petshop with some pets in it:

```
use petshop;
db.pets.insert({name: "Mikey", species: "Gerbil"});
db.pets.insert({name: "Davey Bungooligan", species: "Piranha"});
db.pets.insert({name: "Suzy B", species: "Cat"});
db.pets.insert({name: "Mikey", species: "Hotdog"});
db.pets.insert({name: "Terrence", species: "Sausagedog"});
db.pets.insert({name: "Philomena Jones", species: "Cat"});
```

- Add another piranha, and a naked mole rat called Henry.
- Use find to list all the pets. Find the ID of Mikey the Gerbil.
- Use find to find Mikey by id.
- Use find to find all the gerbils.
- Find all the creatures named Mikey.
- Find all the creatures named Mikey who are gerbils.
- Find all the creatures with the string "dog" in their species.

Procedure

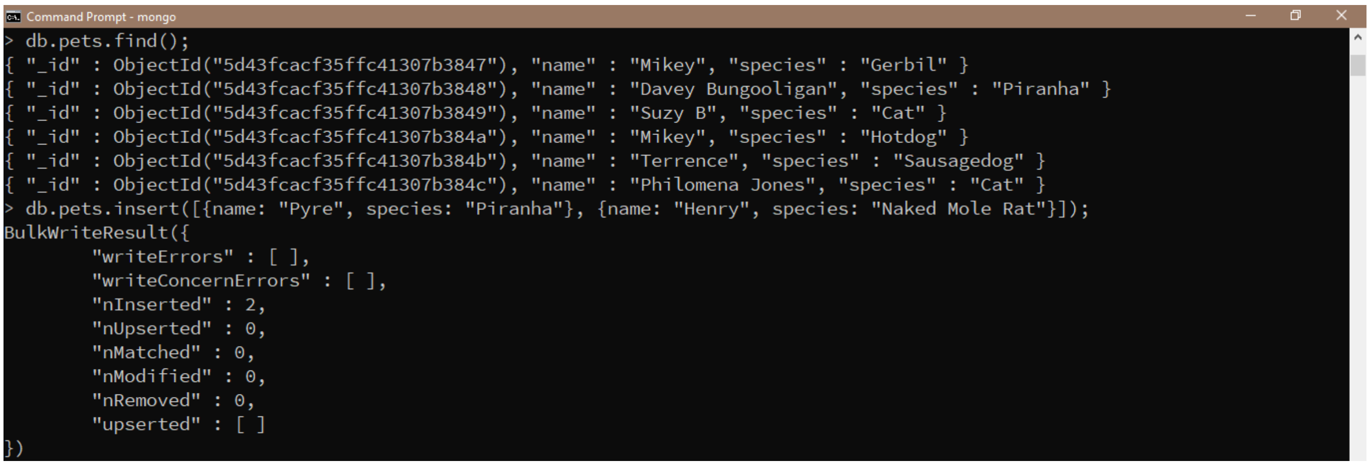
After inserting the specified data, here's how to perform each of the specified commands in MongoDB:

1. Insert the two new pets using **'db.pets.insert([{ name: "Pyre", species: "Piranha" }, { name: "Henry", species: "Naked Mole Rat" }]);'**.
2. Find all pets with **'db.pets.find( );'** and note down the ID of Mikey the Gerbil, which in this case is **'ObjectId("5d43fcacf35ffc41307b3847")'**.
3. Find Mikey by ID using the above ID with **'db.pets.find({ _id: ObjectId ("5d43fcacf35ffc41307b3847") });'**.
4. Find all the Gerbils in the collection using **'db.pets.find({ species: "Gerbil" });'**.
5. Find all the pets named Mikey in the collection with **'db.pets.find({ name: "Mikey" });'**.

6. Find all the pets named Mikey that are Gerbils, by using **‘db.pets.find({ name: "Mikey", species: "Gerbil" });’**.
7. Find all the Gerbils in the collection using **‘db.pets.find({ species: "Gerbil" });’**.
8. Find all pets with the word ‘dog’ in their species by running this command: **‘db.pets.find({ species: /dog/i });’**, the ‘i’ makes our search case insensitive.

Output

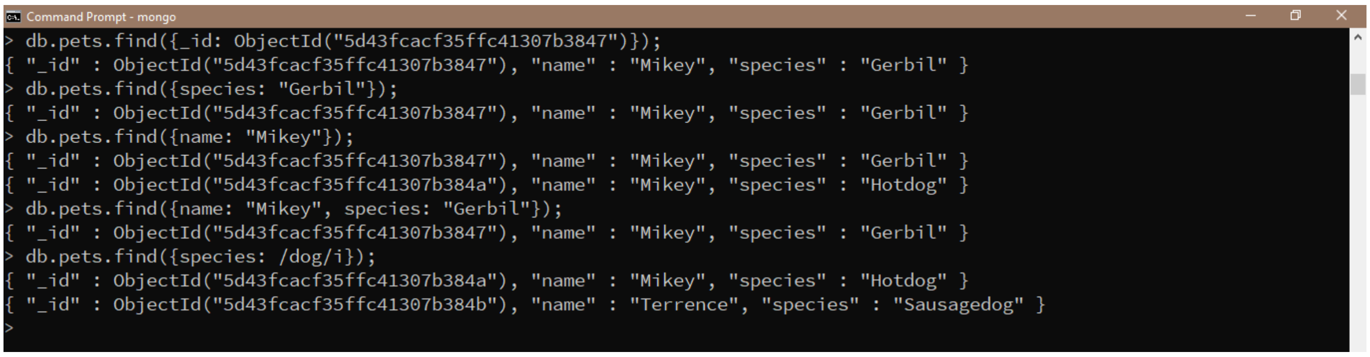
The output after creating the collection, inserting all data is:



Command Prompt - mongo

```
> db.pets.find();
{ "_id" : ObjectId("5d43fcacf35ffc41307b3847"), "name" : "Mikey", "species" : "Gerbil" }
{ "_id" : ObjectId("5d43fcacf35ffc41307b3848"), "name" : "Davey Bungooligan", "species" : "Piranha" }
{ "_id" : ObjectId("5d43fcacf35ffc41307b3849"), "name" : "Suzy B", "species" : "Cat" }
{ "_id" : ObjectId("5d43fcacf35ffc41307b384a"), "name" : "Mikey", "species" : "Hotdog" }
{ "_id" : ObjectId("5d43fcacf35ffc41307b384b"), "name" : "Terrence", "species" : "Sausagedog" }
{ "_id" : ObjectId("5d43fcacf35ffc41307b384c"), "name" : "Philomena Jones", "species" : "Cat" }
> db.pets.insert([{name: "Pyre", species: "Piranha"}, {name: "Henry", species: "Naked Mole Rat"}]);
BulkWriteResult({
        "writeErrors" : [ ],
        "writeConcernErrors" : [ ],
        "nInserted" : 2,
        "nUpserted" : 0,
        "nMatched" : 0,
        "nModified" : 0,
        "nRemoved" : 0,
        "upserted" : [ ]
})
```

The remaining queries will give the output shown below:



Command Prompt - mongo

```
> db.pets.find({_id: ObjectId("5d43fcacf35ffc41307b3847")});
{ "_id" : ObjectId("5d43fcacf35ffc41307b3847"), "name" : "Mikey", "species" : "Gerbil" }
> db.pets.find({species: "Gerbil"});
{ "_id" : ObjectId("5d43fcacf35ffc41307b3847"), "name" : "Mikey", "species" : "Gerbil" }
> db.pets.find({name: "Mikey"});
{ "_id" : ObjectId("5d43fcacf35ffc41307b3847"), "name" : "Mikey", "species" : "Gerbil" }
{ "_id" : ObjectId("5d43fcacf35ffc41307b384a"), "name" : "Mikey", "species" : "Hotdog" }
> db.pets.find({name: "Mikey", species: "Gerbil"});
{ "_id" : ObjectId("5d43fcacf35ffc41307b3847"), "name" : "Mikey", "species" : "Gerbil" }
> db.pets.find({species: /dog/i});
{ "_id" : ObjectId("5d43fcacf35ffc41307b384a"), "name" : "Mikey", "species" : "Hotdog" }
{ "_id" : ObjectId("5d43fcacf35ffc41307b384b"), "name" : "Terrence", "species" : "Sausagedog" }
>
```

EXPERIMENT 06 & 07

Aim (06)

AirPhone Corp is a famous telecom company. They have customers in all locations. Customers use AirPhone Corp's network to make calls. Government has brought in a regulation that all telecom companies should store call details of their customers. This is very important from a security point of view and all telecom companies have to retain this data for 15 years. AirPhone Corp already stores all customer details data, for their analytics team. But due to a surge in mobile users in recent years, their current database cannot handle huge amounts of data. Current database stores only six months of data. AirPhone Corp now wants to scale their database and wants to store 15 years of data.

Data contains following columns – Source: Phone number of caller; Destination: Phone number of call receiver; Source_location: Caller's city; Destination_location: Call receiver's city; Call_duration: phone call duration; Roaming: Flag to check if caller is in roaming; Call_charge: Money charged for call.

Sample Data:
{ source: "+919612345670", destination: "+919612345671", source_location: "Delhi", destination_location: "Mumbai", call_duration: 2.03, roaming: false, call_charge: 2.03 }

After discussing the requirements with database and architecture team, it has been decided that they should use MongoDb. You have been given the task to Setup a distributed system (database) such that data from different locations go to different nodes (to distribute the load):

- Import data to sharded collection
- Check data on each shard for distribution

Aim (07)

Execute below sets of problem by taking reference of Experiment Number 06 and find out:

- Add additional node to existing system (to test if we can add nodes easily when data increases)
- Check the behaviour of cluster (data movement) on adding a shard.
- Check the behaviour of query for finding a document with source location Mumbai.

Procedure

Server creation:

- First we create multiple folders for the servers we're creating through the command prompt, **'C:\>mkdir \dataS\rs1 \dataS\rs2 \dataS\rs3'**.
- We then configure each of them:
  - **'C:\>start mongod --replSet kau --logpath \dataS\rs1\first.log --dbpath \dataS\rs1 --port 27017 --smallfiles --oplogSize 64'**
  - **'C:\>start mongod --replSet kau --logpath \dataS\rs1\first.log --dbpath \dataS\rs1 --port 27017 --smallfiles --oplogSize 64'**
  - **'C:\>start mongod --replSet kau --logpath \dataS\rs1\first.log --dbpath \dataS\rs1 --port 27017 --smallfiles --oplogSize 64'**

Starting the first server:

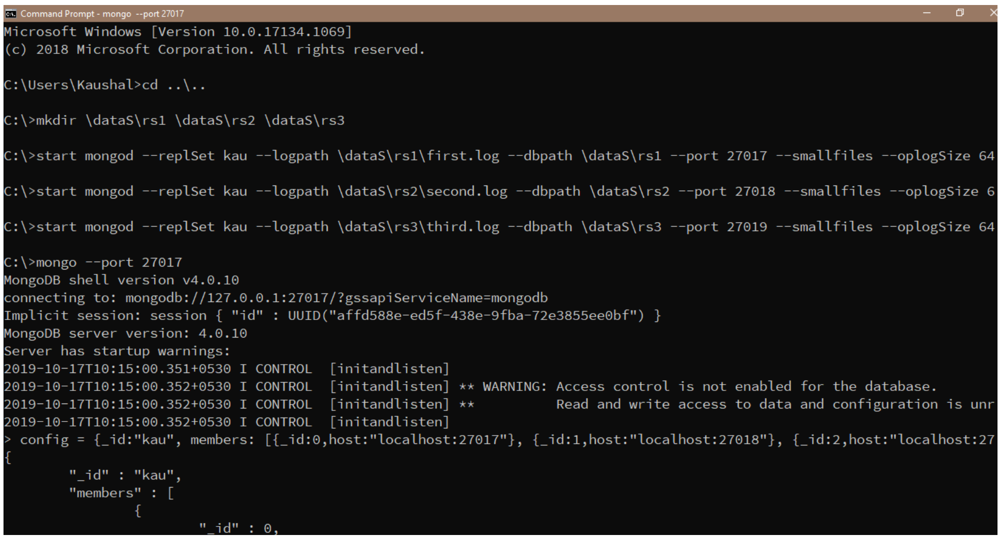
- In the command prompt we then start the first MongoDB server, **'C:\>mongo --port 27017'**, and restart it with **'db.adminCommand({shutdown: 1});'**.
- Then configure the server by setting up configuration attributes with, **'config = { _id: "kau", members: [{ _id: 0, host: "localhost: 27017"}, { _id: 1, host: "localhost: 27018"}, { _id: 2, host: "localhost: 27019"}]};'**, and configure with the command, **'rs.initiate(config);'**.
- This is out PRIMARY server, and we can check its status using **'rs.status();'**.
- After this we insert data into the server, **'db.airphone.insert([{source: "+919612345670", destination: "+919612345671", source_location: "Delhi", destination_location: "Mumbai", call_duration: 2.03, roaming: false, call_charge: 2.03 }, {source: "+919612345670", destination: "+919345371345", source_location: "Delhi", destination_location: "Chennai", call_duration: 1.37, roaming: false, call_charge: 1.37 }, {source: "+919345371345", destination: "+918293977188", source_location: "Chennai", destination_location: "Bangalore", call_duration: 5.55, roaming: false, call_charge: 5.55 }]);'**.
- We can check the data **'db.airphone.find().pretty();'**.

Start the second server:

- In a new command prompt window we start the second MongoDB server, **'mongo --port 27018'**.
- We need to make this the secondary server, which we can do with, **'rs.slaveOk();'**.
- We can now see the data from the primary server on the secondary one, **'db.airphone.find().pretty();'**.

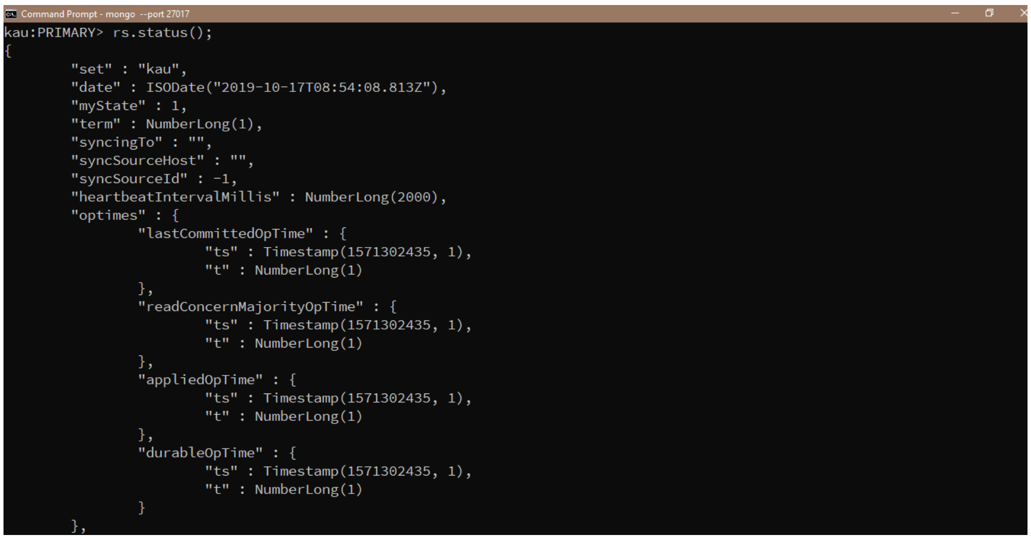
Output

The following are some of the output for the above commands:



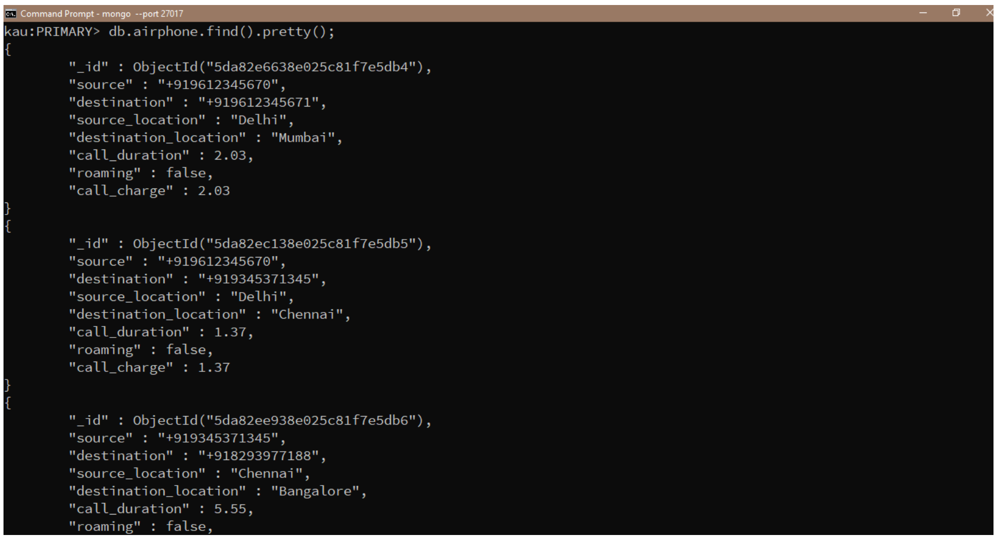
Command Prompt - mongo --port 27017

```
Microsoft Windows [Version 10.0.17134.1069]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Kaushal>cd ..\..

C:\>mkdir \dataS\rs1 \dataS\rs2 \dataS\rs3

C:\>start mongod --replSet kau --logpath \dataS\rs1\first.log --dbpath \dataS\rs1 --port 27017 --smallfiles --oplogSize 64

C:\>start mongod --replSet kau --logpath \dataS\rs2\second.log --dbpath \dataS\rs2 --port 27018 --smallfiles --oplogSize 6

C:\>start mongod --replSet kau --logpath \dataS\rs3\third.log --dbpath \dataS\rs3 --port 27019 --smallfiles --oplogSize 64

C:\>mongo --port 27017
MongoDB shell version v4.0.10
connecting to: mongodb://127.0.0.1:27017/?gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("affd588e-ed5f-438e-9fba-72e3855ee0bf") }
MongoDB server version: 4.0.10
Server has startup warnings:
2019-10-17T10:15:00.351+0530 I CONTROL  [initandlisten]
2019-10-17T10:15:00.352+0530 I CONTROL  [initandlisten] ** WARNING: Access control is not enabled for the database.
2019-10-17T10:15:00.352+0530 I CONTROL  [initandlisten] **          Read and write access to data and configuration is unr
2019-10-17T10:15:00.352+0530 I CONTROL  [initandlisten]
> config = {_id:"kau", members: [{_id:0,host:"localhost:27017"}, {_id:1,host:"localhost:27018"}, {_id:2,host:"localhost:27
{
        "_id" : "kau",
        "members" : [
                {
                        "_id" : 0,
```



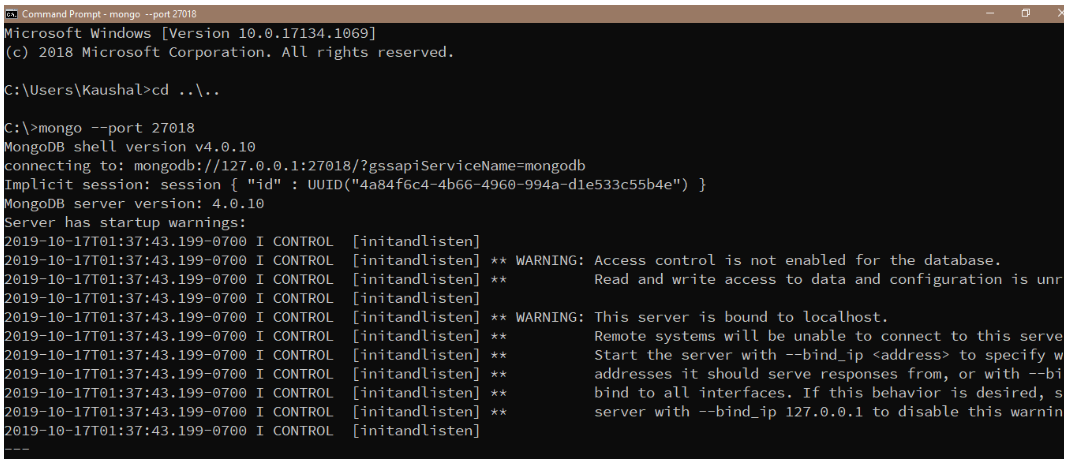
Command Prompt - mongo --port 27017

```
kau:PRIMARY> rs.status();
{
        "set" : "kau",
        "date" : ISODate("2019-10-17T08:54:08.813Z"),
        "myState" : 1,
        "term" : NumberLong(1),
        "syncingTo" : "",
        "syncSourceHost" : "",
        "syncSourceId" : -1,
        "heartbeatIntervalMillis" : NumberLong(2000),
        "optimes" : {
                "lastCommittedOpTime" : {
                        "ts" : Timestamp(1571302435, 1),
                        "t" : NumberLong(1)
                },
                "readConcernMajorityOpTime" : {
                        "ts" : Timestamp(1571302435, 1),
                        "t" : NumberLong(1)
                },
                "appliedOpTime" : {
                        "ts" : Timestamp(1571302435, 1),
                        "t" : NumberLong(1)
                },
                "durableOpTime" : {
                        "ts" : Timestamp(1571302435, 1),
                        "t" : NumberLong(1)
                }
        },
```



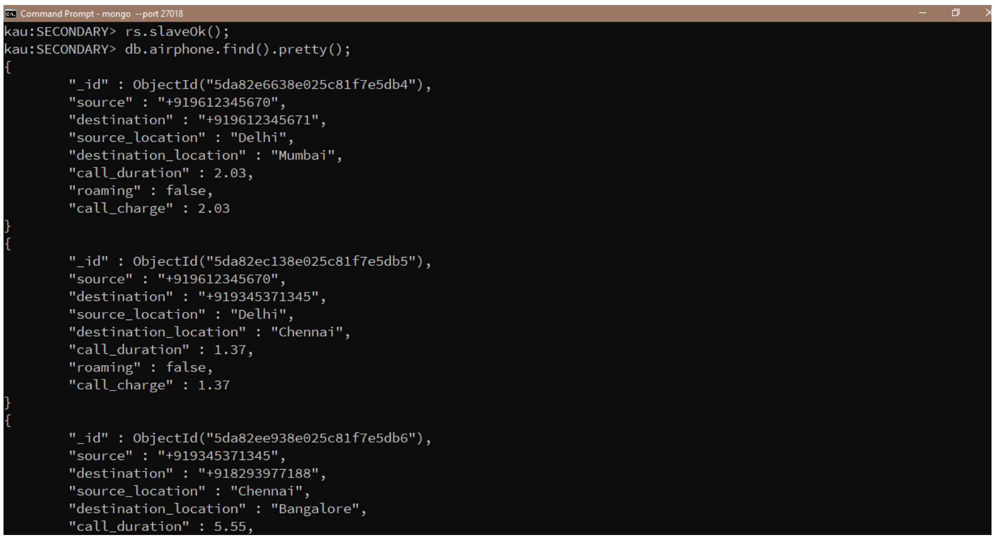
Command Prompt - mongo --port 27017

```
kau:PRIMARY> db.airphone.find().pretty();
{
        "_id" : ObjectId("5da82e6638e025c81f7e5db4"),
        "source" : "+919612345670",
        "destination" : "+919612345671",
        "source_location" : "Delhi",
        "destination_location" : "Mumbai",
        "call_duration" : 2.03,
        "roaming" : false,
        "call_charge" : 2.03
}
{
        "_id" : ObjectId("5da82ec138e025c81f7e5db5"),
        "source" : "+919612345670",
        "destination" : "+919345371345",
        "source_location" : "Delhi",
        "destination_location" : "Chennai",
        "call_duration" : 1.37,
        "roaming" : false,
        "call_charge" : 1.37
}
{
        "_id" : ObjectId("5da82ee938e025c81f7e5db6"),
        "source" : "+919345371345",
        "destination" : "+918293977188",
        "source_location" : "Chennai",
        "destination_location" : "Bangalore",
        "call_duration" : 5.55,
        "roaming" : false,
```



Command Prompt - mongo --port 27018

```
Microsoft Windows [Version 10.0.17134.1069]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Kaushal>cd ..\..

C:\>mongo --port 27018
MongoDB shell version v4.0.10
connecting to: mongodb://127.0.0.1:27018/?gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("4a84f6c4-4b66-4960-994a-d1e533c55b4e") }
MongoDB server version: 4.0.10
Server has startup warnings:
2019-10-17T01:37:43.199-0700 I CONTROL  [initandlisten]
2019-10-17T01:37:43.199-0700 I CONTROL  [initandlisten] ** WARNING: Access control is not enabled for the database.
2019-10-17T01:37:43.199-0700 I CONTROL  [initandlisten] **          Read and write access to data and configuration is unr
2019-10-17T01:37:43.199-0700 I CONTROL  [initandlisten]
2019-10-17T01:37:43.199-0700 I CONTROL  [initandlisten] ** WARNING: This server is bound to localhost.
2019-10-17T01:37:43.199-0700 I CONTROL  [initandlisten] **          Remote systems will be unable to connect to this serve
2019-10-17T01:37:43.199-0700 I CONTROL  [initandlisten] **          Start the server with --bind_ip <address> to specify w
2019-10-17T01:37:43.199-0700 I CONTROL  [initandlisten] **          addresses it should serve responses from, or with --bi
2019-10-17T01:37:43.199-0700 I CONTROL  [initandlisten] **          bind to all interfaces. If this behavior is desired, s
2019-10-17T01:37:43.199-0700 I CONTROL  [initandlisten] **          server with --bind_ip 127.0.0.1 to disable this warnin
2019-10-17T01:37:43.199-0700 I CONTROL  [initandlisten]
---
```



Command Prompt - mongo --port 27018

```
kau:SECONDARY> rs.slaveOk();
kau:SECONDARY> db.airphone.find().pretty();
{
        "_id" : ObjectId("5da82e6638e025c81f7e5db4"),
        "source" : "+919612345670",
        "destination" : "+919612345671",
        "source_location" : "Delhi",
        "destination_location" : "Mumbai",
        "call_duration" : 2.03,
        "roaming" : false,
        "call_charge" : 2.03
}
{
        "_id" : ObjectId("5da82ec138e025c81f7e5db5"),
        "source" : "+919612345670",
        "destination" : "+919345371345",
        "source_location" : "Delhi",
        "destination_location" : "Chennai",
        "call_duration" : 1.37,
        "roaming" : false,
        "call_charge" : 1.37
}
{
        "_id" : ObjectId("5da82ee938e025c81f7e5db6"),
        "source" : "+919345371345",
        "destination" : "+918293977188",
        "source_location" : "Chennai",
        "destination_location" : "Bangalore",
        "call_duration" : 5.55,
```

EXPERIMENT 08

Aim

Anand Corp is a leading corporate training provider. A lot of prestigious organizations send their employees to Anand Corp for training on different skills. As a distinct training provider, Anand Corp has decided to share analysis report with their clients. This report will help their clients know the employees who have completed training and evaluation exam, what are their strengths, and what are the areas where employees need improvement. This is going to be a unique selling feature for the Anand Corp. As Anand Corp is already doing great business and they give training to a large number of people every month, they have huge amount of data to deal with. They have hired you as an expert and want your help to solve this problem.

Attributes of data – Id: id of the person who was trained, Name: name of the person who was trained, Evaluation: evaluation term, Score: score achieved by the person for the specific term.
A person can undergo multiple evaluations. Each evaluation will have a unique result score. You can see the sample data below.

Sample Data
{ "_id": 0, "name": "Andy", "results": [{ "evaluation": "term1", "score": 1.463179736705023 },
{ "evaluation": "term2", "score": 11.78273309957772 }, { "evaluation": "term3", "score": 6.676176060654615 }]}

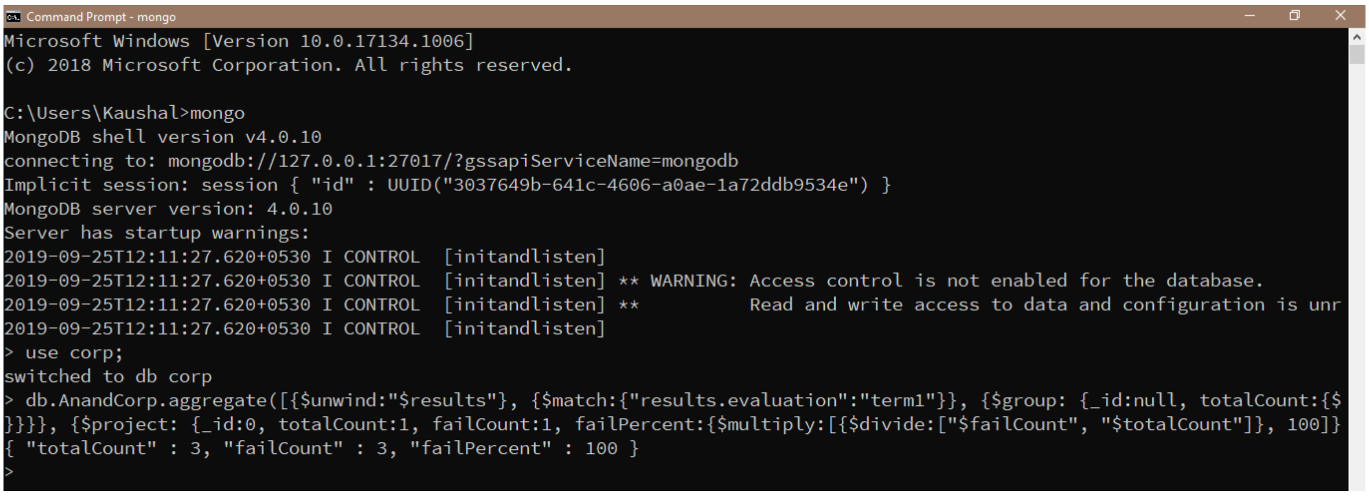
PQR Corp has assigned the following tasks to you to analyze the results:
Find count and percentage of employees who failed in term 1, the passing score being 37.

Procedure

After inserting the specified data and other similar entries, here's how to find count and percentage of employees who failed in term 1, **'db.AnandCorp.aggregate([{ $unwind:"$results"}, { $match: { "results.evaluation": "term1"} },{ $group: { _id: null, totalCount: { $sum:1}, failCount: { $sum: { "$cond": [{ "$lt": ["$results.score", 37] }, 1, 0] } } } }, { $project: { _id: 0, totalCount: 1, failCount: 1, failPercent: { $multiply: [{ $divide: ["$failCount", "$totalCount"] }, 100] } } }]);'**.

Output

The output for the command is:



```
Microsoft Windows [Version 10.0.17134.1006]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Kaushal>mongo
MongoDB shell version v4.0.10
connecting to: mongodb://127.0.0.1:27017/?gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("3037649b-641c-4606-a0ae-1a72ddb9534e") }
MongoDB server version: 4.0.10
Server has startup warnings:
2019-09-25T12:11:27.620+0530 I CONTROL  [initandlisten]
2019-09-25T12:11:27.620+0530 I CONTROL  [initandlisten] ** WARNING: Access control is not enabled for the database.
2019-09-25T12:11:27.620+0530 I CONTROL  [initandlisten] **          Read and write access to data and configuration is unr
2019-09-25T12:11:27.620+0530 I CONTROL  [initandlisten]
> use corp;
switched to db corp
> db.AnandCorp.aggregate([{$unwind:"$results"}, {$match:{"results.evaluation":"term1"}}, {$group: {_id:null, totalCount:{$
}}}}, {$project: {_id:0, totalCount:1, failCount:1, failPercent:{$multiply:[{$divide:["$failCount", "$totalCount"]}, 100]}
{ "totalCount" : 3, "failCount" : 3, "failPercent" : 100 }
>
```

EXPERIMENT 09

Aim

Execute below sets of problem by taking reference of Experimkent Number 08 and find out:

- Find employees who failed in aggregate (term1 + term2 + term3).
- Find the average score of trainees for term1.

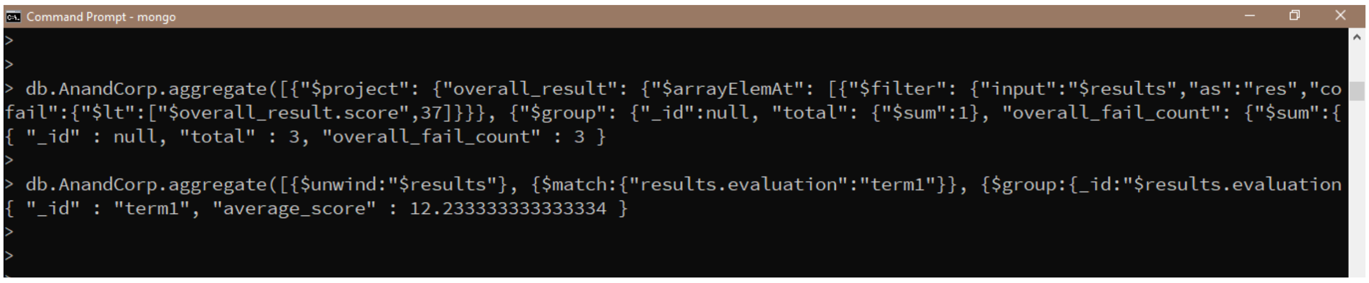
Procedure

After inserting the specified data and other similar entries, here's how to perform the given commands:

1. To find employees who failed in aggregate **'db.AnandCorp.aggregate([{ "$project": { "overall_result": { "$arrayElemAt": [{ "$filter": { "input": "$results", "as": "res", "cond": { "$eq": ["$$res", "evaluation"] } } }, 0] } } }, {"$project": { "overall_fail": { "$lt": ["$overall_result.score", 37] } } }, { "$group": { "_id": null, "total": { "$sum": 1 }, "overall_fail_count": { "$sum": { "$cond": ["overall_fail", 1, 0] } } } }]);'**.
2. To find the average score of trainees for term 1, **'db.AnandCorp.aggregate([{ $unwind: "$results" }, { $match: { "results.evaluation": "term1" } }, { $group: { _id: "$results.evaluation", average_score: { $avg: "$results.score" } } }]);'**.

Output

The output for the commands is:



```
>
>
> db.AnandCorp.aggregate([{"$project": {"overall_result": {"$arrayElemAt": [{"$filter": {"input":"$results","as":"res","co
fail":{"$lt":["$overall_result.score",37]}}}, {"$group": {"_id":null, "total": {"$sum":1}, "overall_fail_count": {"$sum":{
{ "_id" : null, "total" : 3, "overall_fail_count" : 3 }
>
> db.AnandCorp.aggregate([{$unwind:"$results"}, {$match:{"results.evaluation":"term1"}}, {$group:{_id:"$results.evaluation
{ "_id" : "term1", "average_score" : 12.233333333333334 }
>
>
```

EXPERIMENT 10

Aim

Execute below sets of problem by taking reference of Experiment Number 08 and find out:

- Find the average score of trainees for aggregate (term1 + term2 + term3).
- Find number of employees who failed in all the three (term1 + term2 + term3).
- Find the number of employees who failed in any of the three (term1 + term2 + term3).

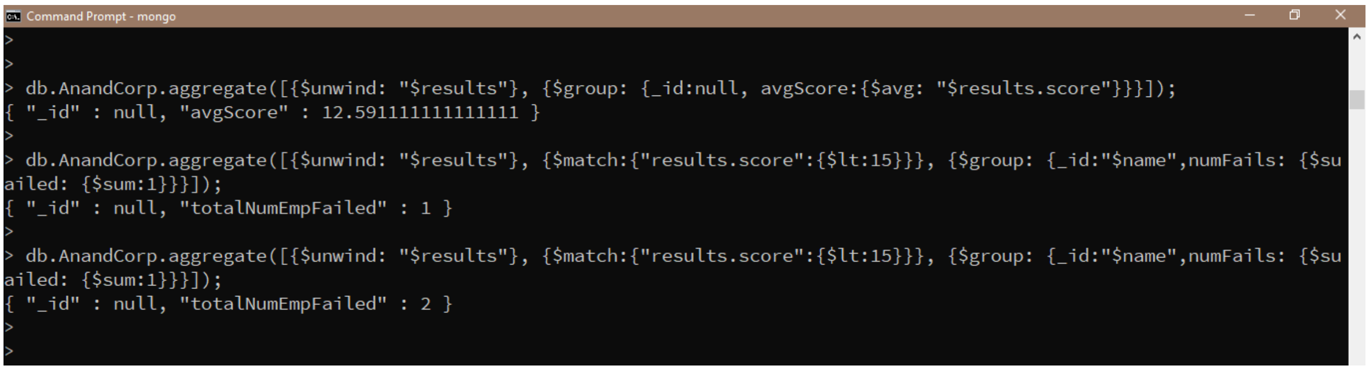
Procedure

After inserting the specified data and other similar entries, here's how to perform the given commands:

1. To find the average score of trainees for aggregate is **'db.AnandCorp.aggregate([{ $unwind: "$results" }, { $group: { _id: null, avgScore: { $avg: "$results.score" } } }]);'**.
2. To find the numbers of employees who failed in all three evaluations, **'db.AnandCorp.aggregate([{ $unwind: "$results" }, { $match: { "results.score": { $lt: 37 } } }, { $group: { _id: "$name", numFails: { $sum: 1 } } }, { $match: { numFails: { $eq: 3 } } }, { $group: { _id: null, totalNumEmpFailed: { $sum: 1 } } }]);'**.
3. The numbers of employees who failed in any of the three evaluations, **'db.AnandCorp.aggregate([{ $unwind: "$results" }, { $match: { "results.score": { $lt: 37 } } }, { $group: { _id: "$name", numFails: { $sum: 1 } } }, { $match: { numFails: { $gt: 0 } } }, { $group: { _id: null, totalNumEmpFailed: { $sum: 1 } } }]);'**.

Output

The output for the commands is:



```
>
>
> db.AnandCorp.aggregate([{$unwind: "$results"}, {$group: {_id:null, avgScore:{$avg: "$results.score"}}}]);
{ "_id" : null, "avgScore" : 12.591111111111111 }
>
> db.AnandCorp.aggregate([{$unwind: "$results"}, {$match:{"results.score":{$lt:15}}}, {$group: {_id:"$name",numFails: {$su
ailed: {$sum:1}}}]);
{ "_id" : null, "totalNumEmpFailed" : 1 }
>
> db.AnandCorp.aggregate([{$unwind: "$results"}, {$match:{"results.score":{$lt:15}}}, {$group: {_id:"$name",numFails: {$su
ailed: {$sum:1}}}]);
{ "_id" : null, "totalNumEmpFailed" : 2 }
>
>
```

EXPERIMENT 11

Aim

Case study on 5 different IT Companies who are working on Mongo DB. Explain on the below parameters:

- Why moved to NoSQL
- Advantages over NoSQL
- Business Benefits
- Technology Adaptation

Procedure

Craigslist

Craigslist is a popular classifieds and job posting community which serves 570 cities in 50 countries. With 1.5 million new classified ads posted every day, Craigslist must archive billions of records in many different formats, and be able to query and report on these archives at runtime.

Why the Move?

Historically, Craigslist stored its information in a MySQL cluster but the lack of flexibility and management costs became barriers for continued use. The original Craigslist archive application took the existing live database data and copied it to the archive system. This caused lengthy delays because changes to the live database schema needed to be propagated to the archive system.

Advantages of NoSQL

- Built-in scalability that allowed each post and its metadata to be stored as a single document.
- Schema changes on the live database so no costly schema migrations.
- Auto-sharding and high availability eased operational pain points.

Business Benefits

- Enabled horizontal scaling across commodity hardware without having to write and maintain complex, custom sharding code.
- Using auto-sharding, the initial deployment was designed to hold over 5 billion documents and 10TB of data.

Technology Adaptation

Craigslist adapted this technology to work with its existing website so that it could transition seamlessly from the MySQL to the NoSQL (MongoDB) database.

Cisco

In November 2011, Cisco launched WebEx Social, an enterprise collaboration platform designed for today's social, mobile, visual and virtual workforce. Their existing relational database needed a NoSQL extension to power the collaborative workspace.

Why the Move?

With their existing relational database, complex SQL queries against highly normalized schema were time consuming and Cisco had little room to scale horizontally.

Advantages of NoSQL

- Easy to implement document model
- Milliseconds response time
- Highly scalable and high availability

Business Benefits

- Cross-reference users and query embedded lists, functions that were either not possible in their existing database or were costly to perform.
- Cisco sped up reads from 30 seconds in some extreme cases to tens of milliseconds per object and eliminated caching needs in certain cases.

Technology Adaptation

Cisco adapted this technology to be able to constantly evaluate the most effective product roadmap to drive customer success on WebEx Social. So, Cisco is able to focus on rolling out new features quickly and offering flexibility to users.

Electronic Arts (EA)

Electronic Arts' EA Sports FIFA, the world's best-selling sports video game franchise. EA Sports FIFA offers otherwise average athletes the chance to take on and beat the world's best. Throughout Asia one of the most popular ways to play the game is with EA's FIFA Online 3.

Why the Move?

Because EA FIFA Online 3, developed by Spearhead, one of EA's leading development studios, needs to scale to millions of players. Spearhead built the game to run on MongoDB as EA already runs 250 servers using it, spread across 80 shards.

Advantages of NoSQL

- Highly scalable
- Auto-sharding

Business Benefits

- Simplified scaling as new players join
- Auto-sharding helps with distribution of load to different servers

Technology Adaptation

EA adapted this technology to work with the game as it was created giving the game the ability to scale as its player base grew after its release.

Thermo Fisher

Thermo Fisher Scientific is the world leader in serving science, with revenues of more than $24 billion and approximately 70,000 employees globally. They help customers accelerate life sciences research, solve complex analytical challenges, improve patient diagnostics, deliver medicines to market and increase laboratory productivity.

Why the Move?

Thermo Fisher is pushing applications to MongoDB Atlas, citing its ease of use, the seamless migration process, and how there has been no downtime, even when reconfiguring the cluster. They began testing MongoDB Atlas around its release date in early July and began launching production applications on the service in September.

Advantages of NoSQL

- High performance
- High flexibility

Business Benefits

- Ability to improve developer productivity
- Ability to be deployed in any environment, cloud or on premises

Technology Adaptation

Thermo Fisher adapted this technology to its spectrometer tool as was able to cut down on the insertion time as shown below:

Database	Milliseconds	Lines of Code
MySQL (not optimized)	147,600 (~2.5 minutes)	21
MySQL (optimized)	410	40
MongoDB (NoSQL)	68	1

GAP

The Gap, Inc., commonly known as Gap Inc. or Gap, (stylized as GAP) is an American worldwide clothing and accessories retailer. In the competitive world of retail, that "instinct" has been set to fast forward as Gap seeks to outpace fast-fashion retailers and other trends that constantly push Gap and other retailers to meet consumer needs, faster.

Why the Move?

Gap's purchase order management system really, really matters in ensuring it can quickly evolve to meet consumer tastes. Unable to meet business agility requirements using traditional relational databases, Gap uses MongoDB for a wide range of supply chain systems, including various master data management, inventory and logistics functions, including purchase order management.

Advantages of NoSQL

- Easily make sense of data that comes in different shapes
- Stores quickly and transparently

Business Benefits

- Power single or individual services and connect them together
- Enables developers to work fast and smart

Technology Adaptation

GAP adapted this technology as it was able to develop this new MongoDB-based purchase order system in just 75 days, a record for the company. In true agile fashion, MongoDB enables Gap to continue to iterate on the system.