SRI KRISHNA ARTS AND SCIENCE COLLEGE

COIMBATORE – 641008

RECORD NOTE

DEPARTMENT : COMPUTER TECHNOLOGY & DATA SCIENCE

NAME	ROLL NUMBER
SUDHARSHAN VIJAY SK	21BDS052

PROGRAMME : BSc Data Science **CLASS : III BSc Data Science**

COURSE : Next Generation Databases - NoSQL

SRI KRISHNA ARTS AND SCIENCE COLLEGE

COIMBATORE – 641008

Roll No.: 21BDS052

Certified bonafide record of work done _____

during the year 2024

Staff in Charge ***Head of the Department***

Submitted to Sri Krishna Arts and Science College (Autonomous), End semester

Examinations held on _____

Internal Examiner ***External Examiner***

DECLARATION

I_____hereby declare that this record of observations is based on the experiments carried out and recorded by me during the laboratory classes of "_____" conducted by **SRI KRISHNA ARTS AND SCIENCE COLLEGE**, Coimbatore-641 008.

Date: ***Signature of the Student***

Name of the Student:

Roll Number:

Countersigned by Staff

LIST OF EXPERIMENTS

Ex.no	Date	Title	Pg.No	Signature
1		Demonstrate simple SQL Queries		
2		Create and Insert Database into MongoDB application		
3		Create MongoDB Query Document		
4		Demonstrate a Query Manipulation in Mongo DB		
5		Demonstrate a Table cloning with Tables		
6		Create JSON File		
7		Search the Text		
8		Create a Regular Expression in the Table		
9		Demonstrate a basic operation on the Document		
10		Demonstrate MongoDB Replication		
11		Create a MongoDB Indexing		

Ex.No:1
Prog .name: **SIMPLE SQL QUERIES**
Date:

AIM:

To Demonstrate a simple SQL Queries

ALGORITHM:

Step 1: Open Run SQL Command Line

Step 2: Establish connection using username ‘system’ and password ‘oracle’

Step 3: Create a database.

Step 4: Create a table and insert few values using INSERT INTO query.

Step 5: Perform various DDL, DML operations in the above created table.

Step 6: Display the output.

Step 7: Stop the program.

SOURCE CODE:

```
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Department VARCHAR(50),
    Salary DECIMAL(10, 2)
);

INSERT INTO Employees (EmployeeID, FirstName, LastName, Department,
Salary)
VALUES (1, 'John', 'Doe', 'HR', 50000),
       (2, 'Jane', 'Smith', 'IT', 60000),
       (3, 'Alice', 'Johnson', 'Finance', 55000),
       (4, 'Bob', 'Williams', 'Marketing', 52000);

SELECT * FROM Employees;

SELECT * FROM Employees WHERE Department = 'IT';

UPDATE Employees
SET Department = 'IT'
WHERE EmployeeID = 5;

DELETE FROM Employees
WHERE EmployeeID = 5;
```

OUTPUT:

```
mysql> use company;
Database changed
mysql> CREATE TABLE Employees (
    ->     EmployeeID INT PRIMARY KEY,
    ->     FirstName VARCHAR(50),
    ->     LastName VARCHAR(50),
    ->     Department VARCHAR(50),
    ->     Salary DECIMAL(10, 2)
    -> );
Query OK, 0 rows affected (0.02 sec)

mysql> INSERT INTO Employees (EmployeeID, FirstName, LastName, Department, Salary)
    -> VALUES (1, 'John', 'Doe', 'HR', 50000),
    ->        (2, 'Jane', 'Smith', 'IT', 60000),
    ->        (3, 'Alice', 'Johnson', 'Finance', 55000),
    ->        (4, 'Bob', 'Williams', 'Marketing', 52000);
Query OK, 4 rows affected (0.02 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM Employees;
+------------+-----------+----------+------------+----------+
| EmployeeID | FirstName | LastName | Department | Salary   |
+------------+-----------+----------+------------+----------+
|          1 | John      | Doe      | HR         | 50000.00 |
|          2 | Jane      | Smith    | IT         | 60000.00 |
|          3 | Alice     | Johnson  | Finance    | 55000.00 |
|          4 | Bob       | Williams | Marketing  | 52000.00 |
+------------+-----------+----------+------------+----------+
4 rows in set (0.00 sec)

mysql> SELECT * FROM Employees WHERE Department = 'IT';
+------------+-----------+----------+------------+----------+
| EmployeeID | FirstName | LastName | Department | Salary   |
+------------+-----------+----------+------------+----------+
|          2 | Jane      | Smith    | IT         | 60000.00 |
+------------+-----------+----------+------------+----------+
1 row in set (0.00 sec)

mysql> UPDATE Employees
    -> SET Department = 'IT'
    -> WHERE EmployeeID = 5;
Query OK, 0 rows affected (0.00 sec)
Rows matched: 0  Changed: 0  Warnings: 0

mysql> DELETE FROM Employees
    -> WHERE EmployeeID = 5;
Query OK, 0 rows affected (0.00 sec)
```

RESULT:

Hence the above program has been compiled and executed successfully.

Ex.No:2
Prg.Name: **CREATE & INSERT DB**
Date:

AIM:

To Create and Insert Database into MongoDB Application

ALGORITHM:

Step 1: Open MongoDB Compass.

Step 2: Use mongosh inside the MongoDB Compass.

Step 3: Create a database using USE command.

Step 4: Create a collection using createCollection() function.

Step 5: Insert documents in JSON format inside the collection using insertOne() and insertMany() function.

Step 6: Display the output using find() function.

Step 7: Stop the program.

SOURCE CODE:

Use Hotstar

db.createCollection(“Movies”)

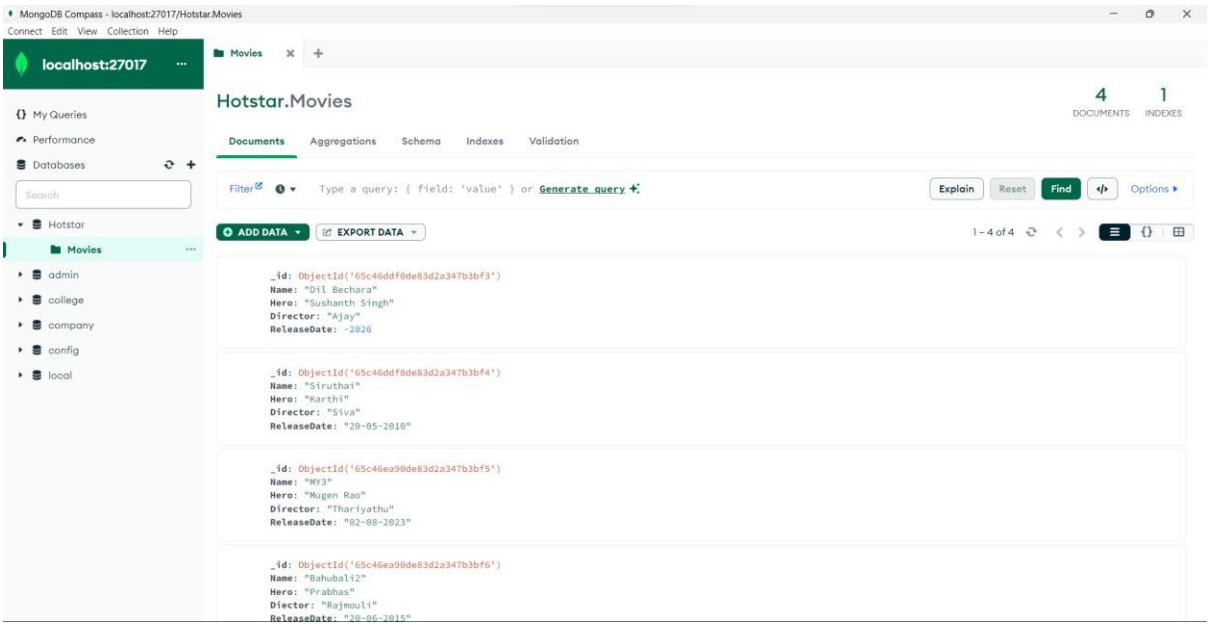
db.Movies.insertOne({Name: "Good Night", Hero: "Manikandan", Director: "Kumar", ReleaseDate: "01-06-2023”})

Hotstar> db.Movies.insertMany([{Name: "HY3", Hero: "MugenRao", Director: "Thariyathu", ReleaseDate: 02-08-2023"}], [{Name: "Bahubali2", Hero: "Prabhas, Diector: "Rajmouli", ReleaseDate: 20-06-2015"}])

OUTPUT:

```
>_MONGOSH

> use Hotstar
< switched to db Hotstar
> db.Movies.insertOne({Name:"Good Night",Hero:"Manikandan",Director:"Kumar",ReleaseDate:"01-06-2023"})
< {
    acknowledged: true,
    insertedId: ObjectId('65c46ff418943904390490fc')
  }
Hotstar>
```

```
Hotstar> db.Movies.insertOne({Name:"Hanuman",Hero:"Aathi",Director:"Raghul",ReleaseDate:"01-01-2024"})
{
  acknowledged: true,
  insertedId: ObjectId('65c46f980de83d2a347b3bf9')
}
Hotstar> db.Movies.insertMany([{Name:"MY3",Hero:"Mugen Rao",Director:"Thariyathu",ReleaseDate:"02-08-2023"},{Name:"Bahubali2",Hero:"Prabhas",Diector:"Rajmou
li",ReleaseDate:"20-06-2015"}])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId('65c470080de83d2a347b3bfa'),
    '1': ObjectId('65c470080de83d2a347b3bfb')
  }
}
Hotstar>
```



RESULT:

Hence the above program has been compiled and executed successfully

Ex.No:3
Prg.Name: **MONGODB QUERY DOCUMENT**
Date:

AIM:

To Create MongoDB Query Document

ALGORITHM:

Step 1: Open MongoDB Compass.

Step 2: Use mongosh inside the MongoDB Compass.

Step 3: Specify the database and collection you want to query within the connected MongoDB instance.

Step 4: Create a query object to define the criteria for your search.

Step 5: Use MongoDB query operators like $eq, $gt, $lt, limit, sort etc.

Step 6: Execute the query against the specified collection using the find() method

Step 7: Retrieve the query results, which may include one or multiple documents that match the specified criteria.

Step 8: Stop the process.

SOURCE CODE:

Use Hotstar

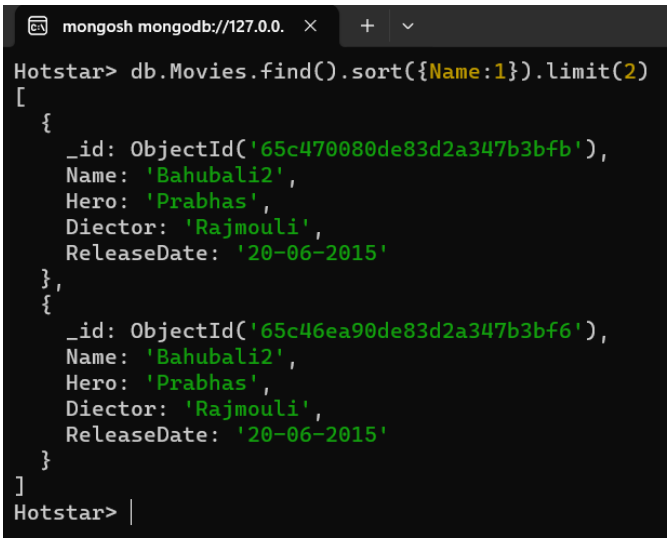
db.Movies.insertOne({Name: "Good Night", Hero: "Manikandan", Director: "Kumar", ReleaseDate: "01-06-2023”})

Hotstar> db.Movies.insertMany([[Name: "HY3", Hero: "MugenRao", Director: "Thariyathu", ReleaseDate: 02-08-2023"], [Name: "Bahubali2", Hero: "Prabhas, Diector: "Rajmouli", ReleaseDate: 20-06-2015"}])

db.Movies.find().sort({Name:1}).limit(2)

db.Movies.find().pretty()

OUTPUT:



RESULT:

Hence the above program has been compiled and executed successfully.

Ex.no:4
Prg.name: **QUERY MANIPULATION**
Date:

AIM:

Demonstrate Query Manipulation in MongoDB

ALGORITHM:

Step 1: Open MongoDB Compass.

Step 2: Use mongosh inside the MongoDB Compass.

Step 3: Specify the database and collection you want to query within the connected MongoDB instance.

Step 4: Use the find() method to retrieve documents based on specified criteria.

Step 5: Use the insertOne() or insertMany() methods to add new documents to the collection.

Step 6: Use the insertOne() or insertMany() methods to add new documents to the collection.

Step 7: Use the deleteOne() or deleteMany() methods to remove documents that match specified filters.

Step 8: Display the output using find() function.

Step 9: Stop the program.

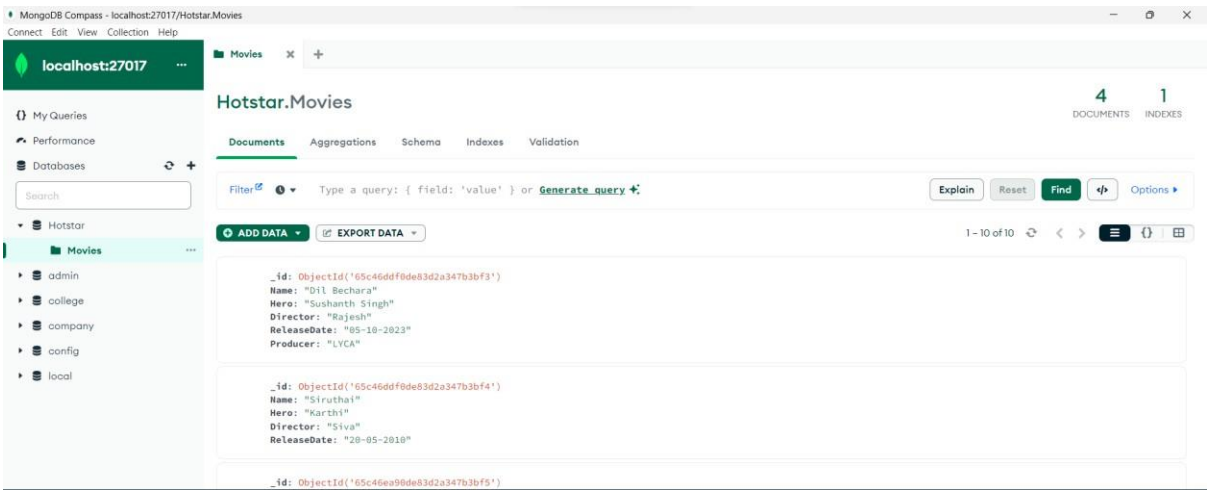
SOURCE CODE:

```
db.Movies.updateOne({Name: "Dil Bechara"}, {$set: {Director: "Rajesh"}})

db.Movies.replaceOne({}, {Name: "Dil Bechara", Hero: "Sushanth Singh",
Director: "Ajay", ReleaseDate: "05-10-2023", Producer: "LYCA"})

db.Movies.deleteOne({ Name: "Dil Bechara" });

db.Movies.deleteMany({ Director: "Rajesh" });
```

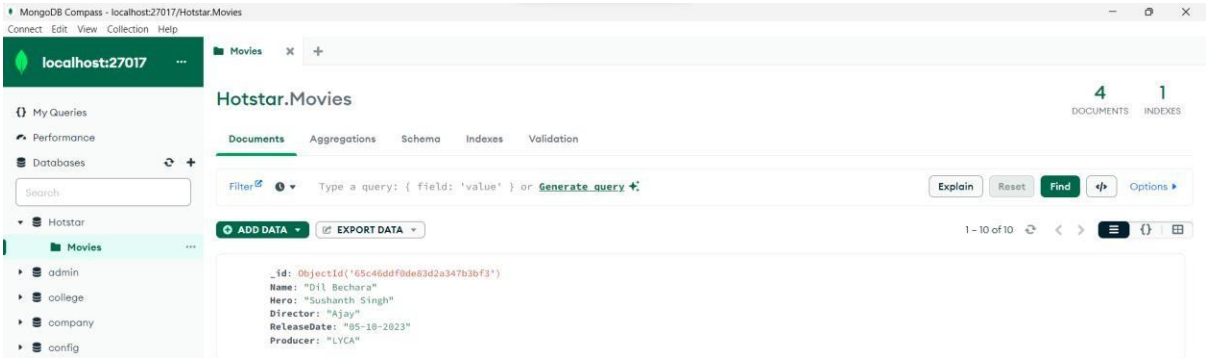
OUTPUT:

```
> db.Movies.updateOne({Name:"Dil Bechara"},{$set:{Director:"Rajesh"}})
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0
  }
> db.Movies.replaceOne({},{Name:"Dil Bechara",Hero:"Sushanth Singh",Director:"Ajay",ReleaseDate:"05-10-2023",Producer:"LYCA"})
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0
  }
Hotstar>
```

Query 1:



Query 2:



RESULT:

Hence the above program has been compiled and executed successfully.

Ex.no:5
Prg.Name: **TABLE CLONING**
Date:

AIM:

Demonstrate a Table cloning with Tables

ALGORITHM:

Step 1: Open MongoDB Compass.

Step 2: Create a database along with two collections.

Step 3: Insert a document inside of the first collection.

Step 4: Export the same document into JSON format and save it.

Step 5: Now, escalate to the second collection, using add data, import the saved JSON document here.

Step 6: The file will be successfully imported. Hence, two copies are created, thereby cloning is performed.

Step 7: Stop the process

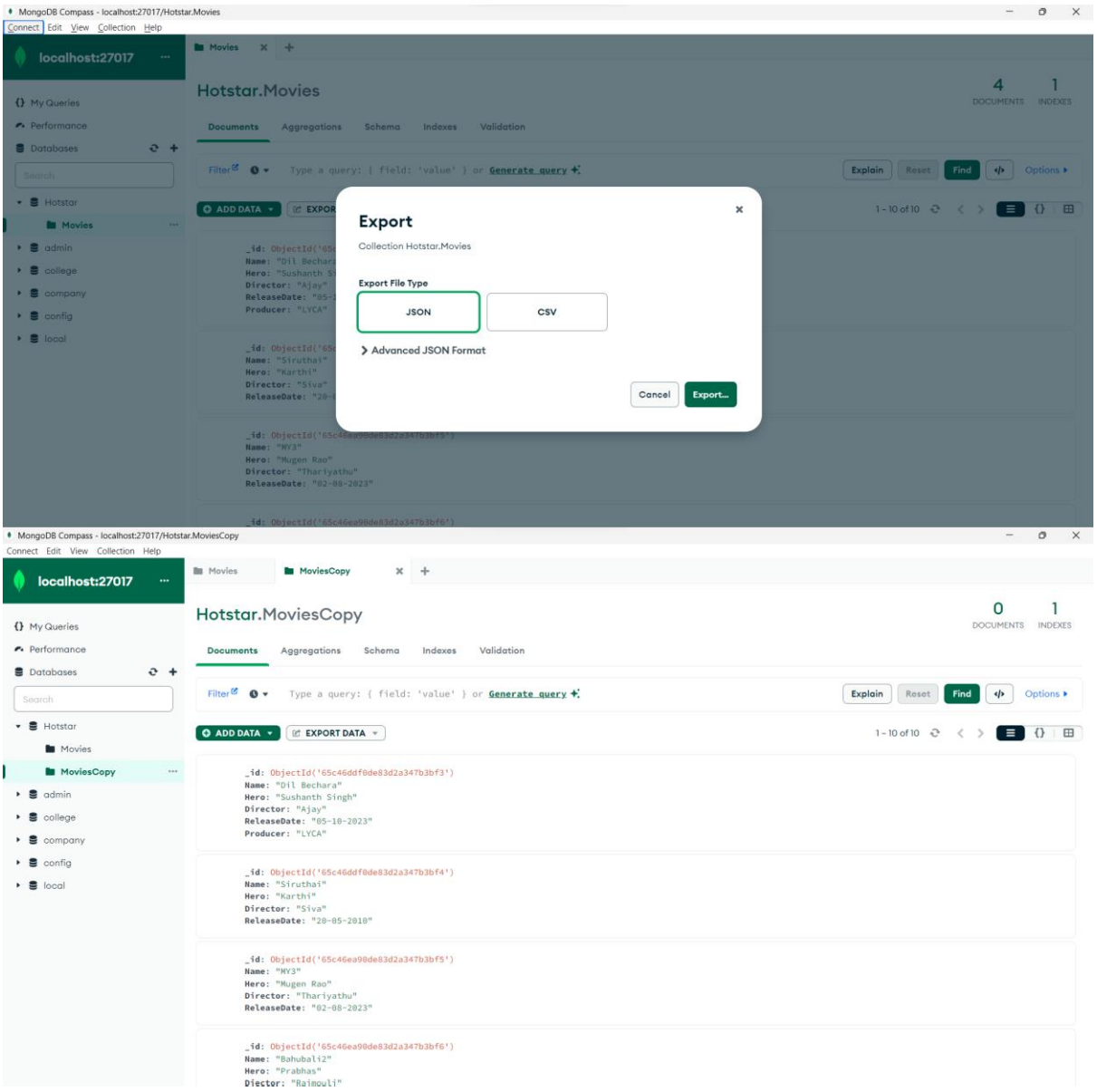
SOURCE CODE:

Export the file to JSON Format

db.createCollection("MoviesCopy")

Click on ADD DATA and load the JSON File

OUTPUT:



RESULT:

Hence the above program has been compiled and executed successfully.

Ex.No:6
Prg.Name: **JSON FILE**
Date:

AIM:

To Create a JSON File and add the file in MongoDB

ALGORITHM:

Step 1: Open MongoDB Compass.

Step 2: Use mongosh inside the MongoDB Compass.

Step 3: Decide on the structure and content of your JSON file.

Step 4: create an empty object {}.

Step 5: Populate the JSON object with key-value pairs representing your data.

Step 6: For nested objects or arrays, create additional objects or arrays within the main JSON object.

Step 7: Choose or create the directory where you want to save the JSON file.

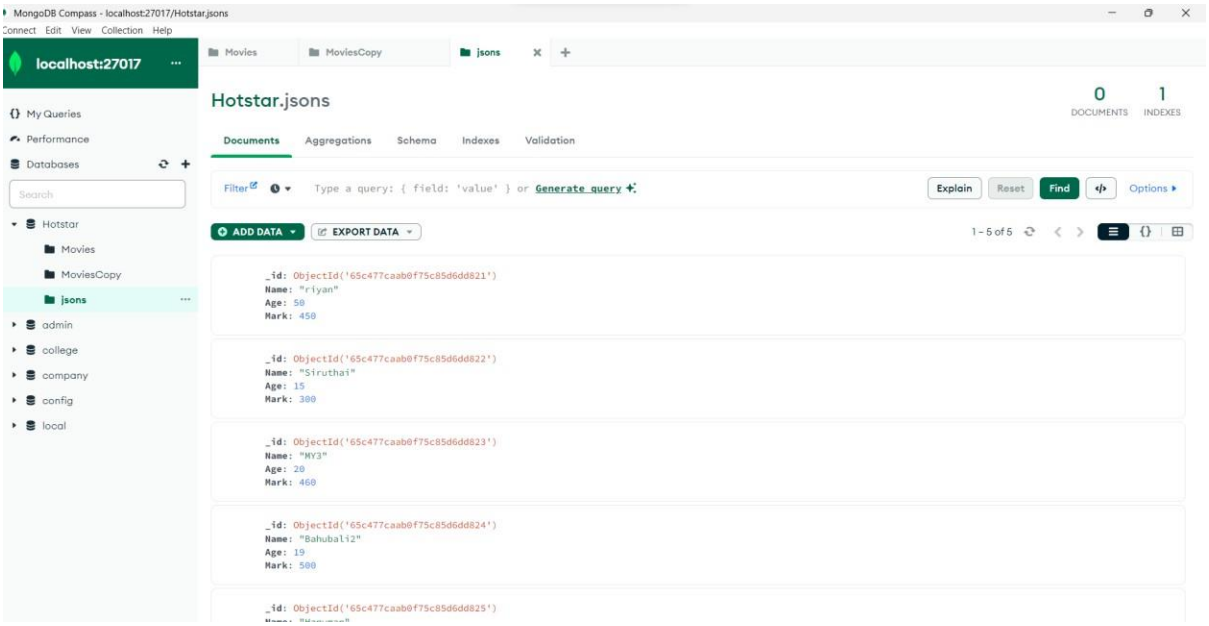
Step 8: Specify the file name with the .json extension.

Step 9: Stop the program

SOURCE CODE:


```
[{
  "Name": "riyan",
  "Age": 50,
  "Mark": 450
},
{
  "Name": "Siruthai",
  "Age": 15,
  "Mark": 300
},
{
  "Name": "MY3",
  "Age": 20,
  "Mark": 460
},
{
  "Name": "Bahubali2",
  "Age": 19,
  "Mark": 500
},
{
  "Name": "Hanuman",
  "Age": 20,
  "Mark": 300
}]
```

OUTPUT:



RESULT:

Hence the above program has been compiled and executed successfully.

Ex.no:7
Prg.Name: **SEARCH THE TEXT**
Date:

AIM:

To Search the Text in MongoDB

ALGORITHM:

Step 1: Open MongoDB Compass.

Step 2: Use mongosh inside the MongoDB Compass.

Step 3: Insert a document of JSON format.

Step 4: Create an index for all the fields inside the document.

Step 5: Search the value using $search keyword.

Step 6: Use find() to display the output.

Step 7: Stop the program.

SOURCE CODE:

```
db.MoviesCopy.createIndex({ Name: "text", Description: "text" }

db.MoviesCopy.find({ $text: {$search: "Good Night" }})

db.MoviesCopy.find({$text: {$search: "siruthai", caseSenstive: true
}})

db.MoviesCopy.find({$text: {$search: "Siruthai", caseSenstive: true
}})
```

OUTPUT:

```
> db.MoviesCopy.createIndex({ Name: "text", Description: "text" })
< Name_text_Description_text
> db.MoviesCopy.find({ $text: { $search: "Good Night" } })
< {
    _id: ObjectId('65c46ff4189439043904490fc'),
    Name: 'Good Night',
    Hero: 'Manikandan',
    Director: 'Kumar',
    ReleaseDate: '01-06-2023'
  }
Hotstar>
```

```
> db.MoviesCopy.find( { $text: { $search: "siruthai", $caseSensitive: true } } )
<
> db.MoviesCopy.find( { $text: { $search: "Siruthai", $caseSensitive: true } } )
< {
    _id: ObjectId('65c46ddf0de83d2a347b3bf4'),
    Name: 'Siruthai',
    Hero: 'Karthi',
    Director: 'Siva',
    ReleaseDate: '20-05-2010'
  }
Hotstar>
```

RESULT:

Hence the above program has been compiled and executed successfully.

Ex.no:8
Prg.Name**:** **REGULAR EXPRESSION**
Date:

AIM:

To Create a Regular Expression in MongoDB Tables.

ALGORITHM:

Step 1: Open MongoDB Compass.

Step 2: Use mongosh inside the MongoDB Compass.

Step 3: Choose the collection where you want to apply the regular expression query.

Step 4: Determine the field within the collection where you want to search for patterns

Step 5: Use the $regex operator to construct your regular expression query.

Step 6: Optionally, apply additional options like case insensitivity or matching multiple lines depending on your requirements.

Step 7: Use find() to display the output.

Step 8: Stop the program.

SOURCE CODE:

```
db.MoviesCopy.find({ Name: {$regex: "Good Night"}}).pretty()

db.MoviesCopy.find({ Name: {$regex: "Siruthai",$options: "i"}}).pretty()
```

OUTPUT:

```
>_MONGOSH
> db.Movies.find({Name:{$regex:"Good Night"}}).pretty()
< {
    _id: ObjectId('65c46ff418943904390490fc'),
    Name: 'Good Night',
    Hero: 'Manikandan',
    Director: 'Kumar',
    ReleaseDate: '01-06-2023'
  }
> db.Movies.find({Name:{$regex:"Siruthai",$options:"i"}}).pretty()
< {
    _id: ObjectId('65c46ddf0de83d2a347b3bf4'),
    Name: 'Siruthai',
    Hero: 'Karthi',
    Director: 'Siva',
    ReleaseDate: '20-05-2010'
  }
  {
    _id: ObjectId('65c481be18943904390490fd'),
    Name: 'siruthai',
    Hero: 'karthi',
    Director: 'siva',
    ReleaseDate: '01-06-2023'
  }
Hotstar>
```

RESULT:

Hence the above program has been compiled and executed successfully.

Ex.no:9
Prg.Name: **BASIC OPERATION IN MONGODB**
Date:

AIM:

Demonstrate Basic operations on the Document

ALGORITHM:

Step 1: Open MongoDB Compass.

Step 2: Use mongosh inside the MongoDB Compass.

Step 3: Use the insertOne() or insertMany() method to add new documents to a collection.

Step 4: Use the find() method to retrieve documents from a collection.

Step 5: Use the updateOne() or updateMany() method to modify existing documents in a collection.

Step 6: Use the deleteOne() or deleteMany() method to remove documents from a collection.

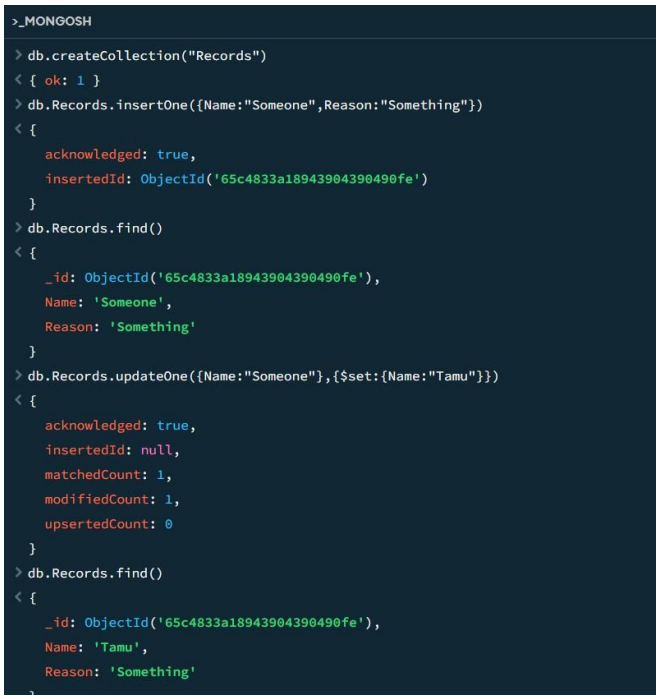
Step 7: Use find() to display the output.

Step 8: Stop the program.

SOURCE CODE:

```
db.createCollection("Records")

db.Records.insertOne({Name:"Someone",Reason:"Something"})

db.Records.find()

db.Records.updateOne({Name: "Someone"} {$set:{Name: "Tamu"}})

db.Records.find()

db.Records.deleteOne({Name: "Someone"})

db.Records.find()
```

OUTPUT:



RESULT:

Hence the above program has been compiled and executed successfully.

Ex.no:10
Prg.Name: **MONGODB REPLICATON**
Date:

AIM:

To Demonstrate MongoDB Replication.

ALGORITHM:

Step 1: Open MongoDB Compass.

Step 2: Use mongosh inside the MongoDB Compass.

Step 3: Open C Drive, create a folder named data. Add 3replica folders to it.

Step 4: Copy the path of bin folder of MongoDB.

Step 5: Open command prompt and change the directory to MongoDB folder.

Step 6: Start the created replica sets.

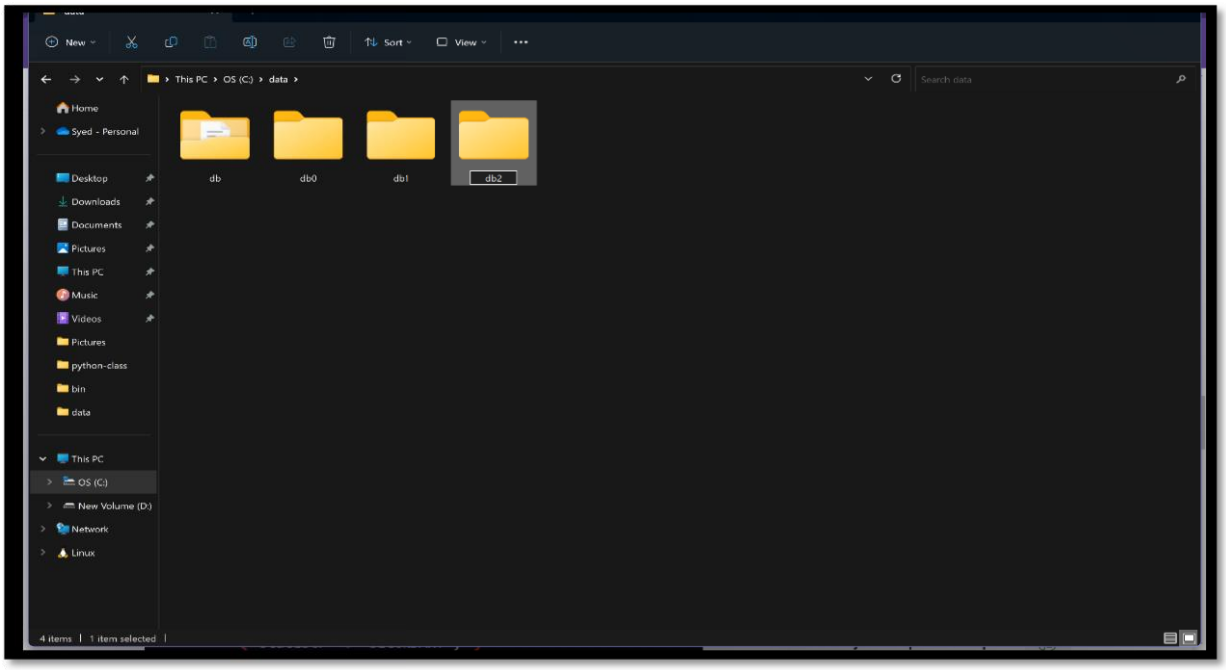
Step 7: Initiate their running using rs.initiate().

Step 6: Use rs.status() to view their status.

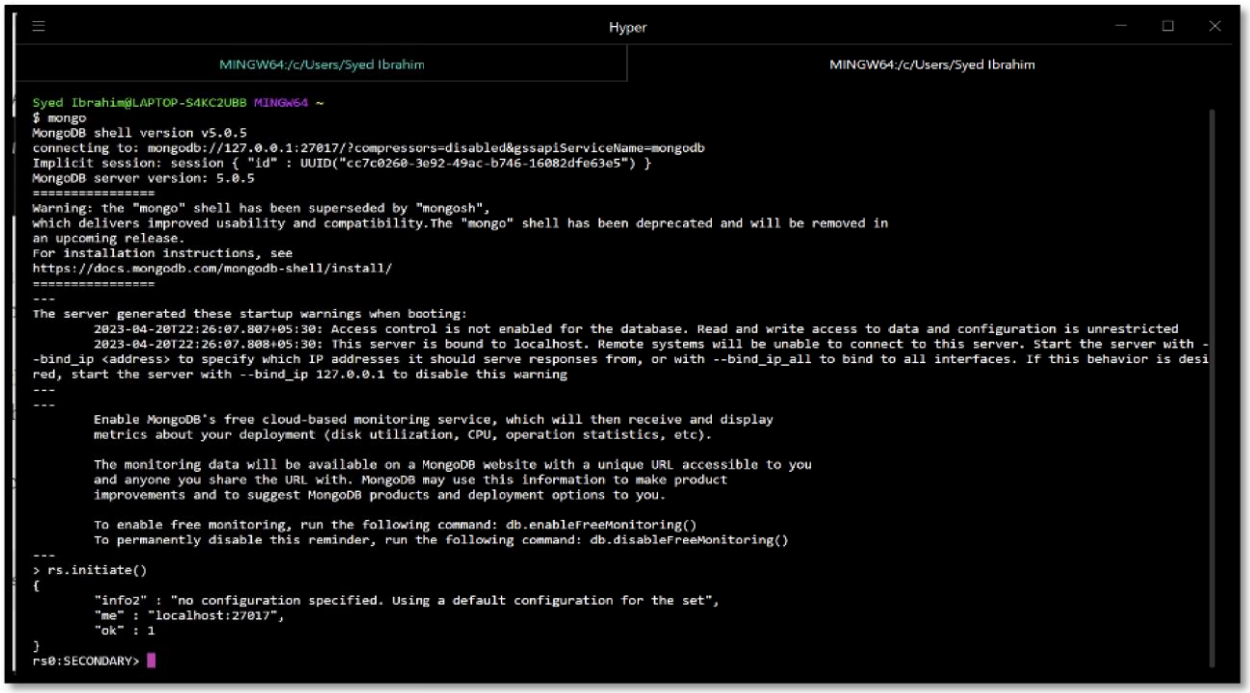
Step 7: Open mongodb compass and

create a Step 8: Stop the program.

PROGRAM:


New
Sort
View
This PC › OS (C:) › data ›
Search data
Home
Syed - Personal
Desktop
Downloads
Documents
Pictures
This PC
Music
Videos
Pictures
python-class
bin
data
This PC
OS (C:)
New Volume (D:)
Network
Linux
db
db0
db1
db2
4 items | 1 item selected

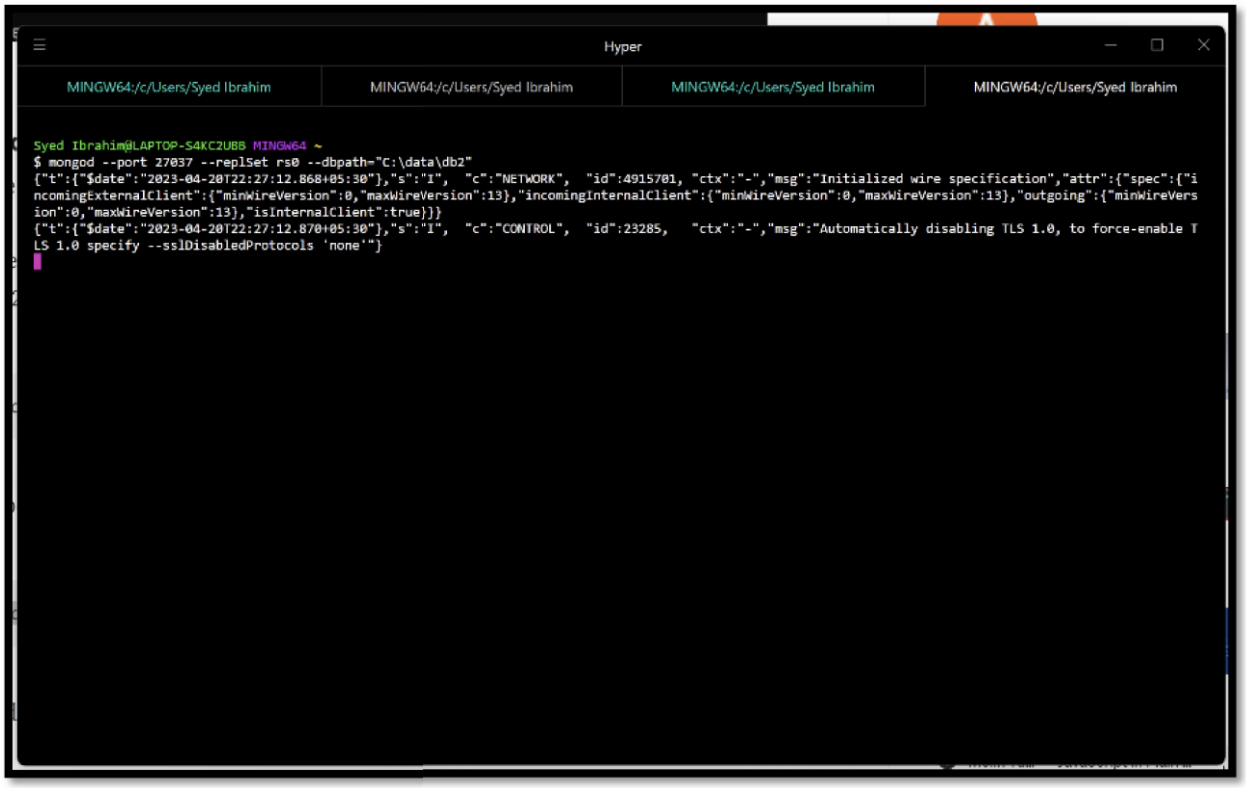

MINGW64:/c/Users/Syed Ibrahim
Syed Ibrahim@LAPTOP-S4KC2UBB MINGW64 ~
$ mongod --port 27017 --replSet rs0 --dbpath="C:\data\db0"
{"t":{"$date":"2023-04-20T22:25:02.789+05:30"},"s":"I", "c":"CONTROL", "id":23285, "ctx":"-","msg":"Automatically disabling TLS 1.0, to force-enable TLS 1.0 specify --sslDisabledProtocols 'none'"}


Hyper
MINGW64:/c/Users/Syed Ibrahim
MINGW64:/c/Users/Syed Ibrahim
Syed Ibrahim@LAPTOP-S4KC2UBB MINGW64 ~
$ mongo
MongoDB shell version v5.0.5
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("cc7c0260-3e92-49ac-b746-16082dfe63e5") }
MongoDB server version: 5.0.5
==========
Warning: the "mongo" shell has been superseded by "mongosh",
which delivers improved usability and compatibility.The "mongo" shell has been deprecated and will be removed in
an upcoming release.
For installation instructions, see
https://docs.mongodb.com/mongodb-shell/install/
==========

The server generated these startup warnings when booting:
2023-04-20T22:26:07.807+05:30: Access control is not enabled for the database. Read and write access to data and configuration is unrestricted
2023-04-20T22:26:07.808+05:30: This server is bound to localhost. Remote systems will be unable to connect to this server. Start the server with -
-bind_ip <address> to specify which IP addresses it should serve responses from, or with --bind_ip_all to bind to all interfaces. If this behavior is desi
red, start the server with --bind_ip 127.0.0.1 to disable this warning
---

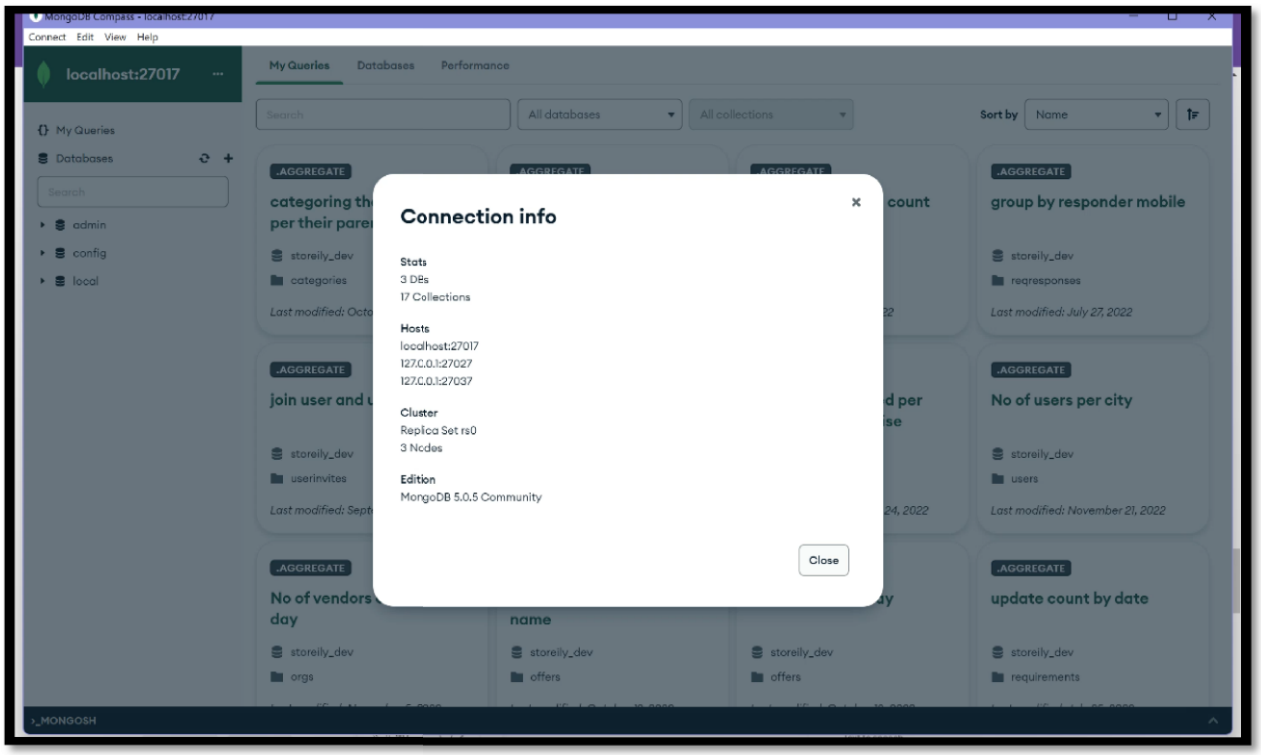
Enable MongoDB's free cloud-based monitoring service, which will then receive and display
metrics about your deployment (disk utilization, CPU, operation statistics, etc).
The monitoring data will be available on a MongoDB website with a unique URL accessible to you
and anyone you share the URL with. MongoDB may use this information to make product
improvements and to suggest MongoDB products and deployment options to you.
To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()

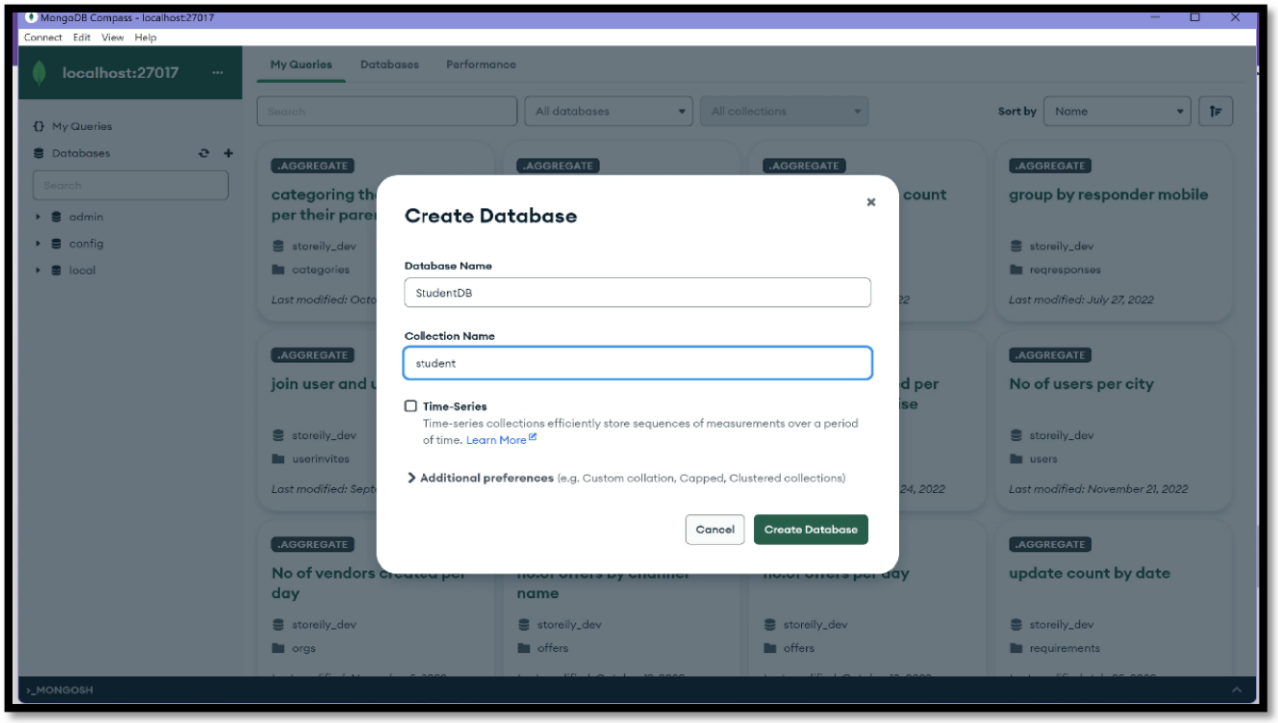
> rs.initiate()
{
"info2" : "no configuration specified. Using a default configuration for the set",
"me" : "localhost:27017",
"ok" : 1
}
rs0:SECONDARY>

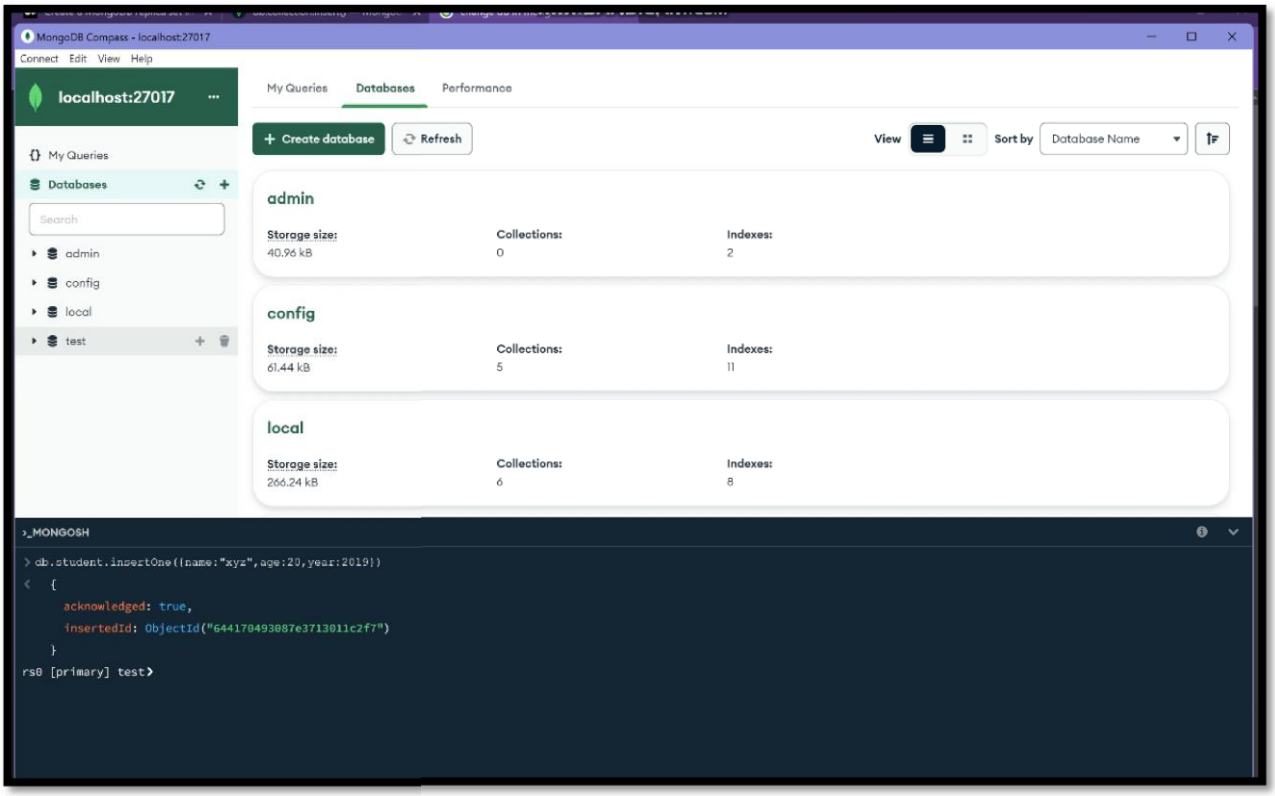

Hyper
MINGW64:/c/Users/Syed Ibrahim
MINGW64:/c/Users/Syed Ibrahim
MINGW64:/c/Users/Syed Ibrahim
MINGW64:/c/Users/Syed Ibrahim
Syed Ibrahim@LAPTOP-S4KC2UBB MINGW64 ~
$ mongod --port 27037 --replSet rs0 --dbpath="C:\data\db2"
{"t":{"$date":"2023-04-20T22:27:12.868+05:30"},"s":"I", "c":"NETWORK", "id":4915701, "ctx":"-","msg":"Initialized wire specification","attr":{"spec":{"i
ncomingExternalClient":{"minWireVersion":0,"maxWireVersion":13},"incomingInternalClient":{"minWireVersion":0,"maxWireVersion":13},"outgoing":{"minWireVers
ion":0,"maxWireVersion":13},"isInternalClient":true}}}
{"t":{"$date":"2023-04-20T22:27:12.870+05:30"},"s":"I", "c":"CONTROL", "id":23285, "ctx":"-","msg":"Automatically disabling TLS 1.0, to force-enable T
LS 1.0 specify --sslDisabledProtocols 'none'"}

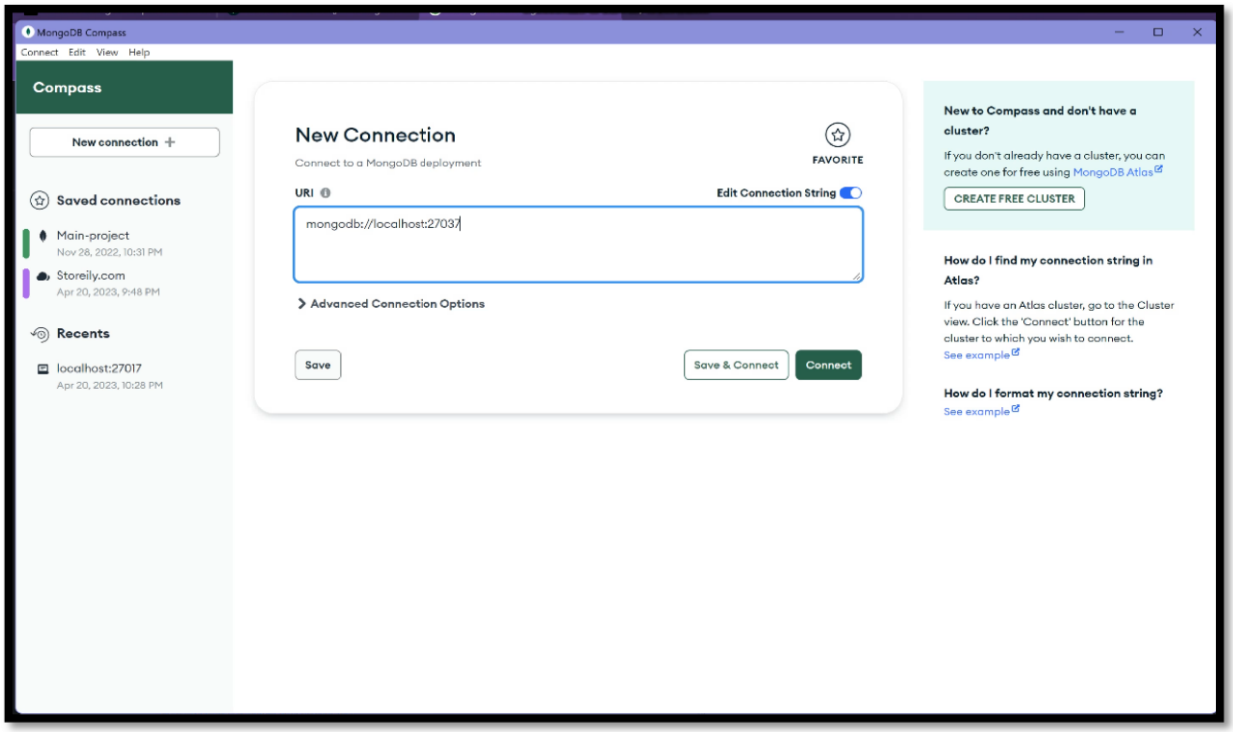

Hyper
MINGW64:/c/Users/Syed Ibrahim
MINGW64:/c/Users/Syed Ibrahim
MINGW64:/c/Users/Syed Ibrahim
MINGW64:/c/Users/Syed Ibrahim
The monitoring data will be available on a MongoDB website with a unique URL accessible to you
and anyone you share the URL with. MongoDB may use this information to make product
improvements and to suggest MongoDB products and deployment options to you.
To enable free monitoring, run the following command: db.enableFreeMonitoring()
To permanently disable this reminder, run the following command: db.disableFreeMonitoring()

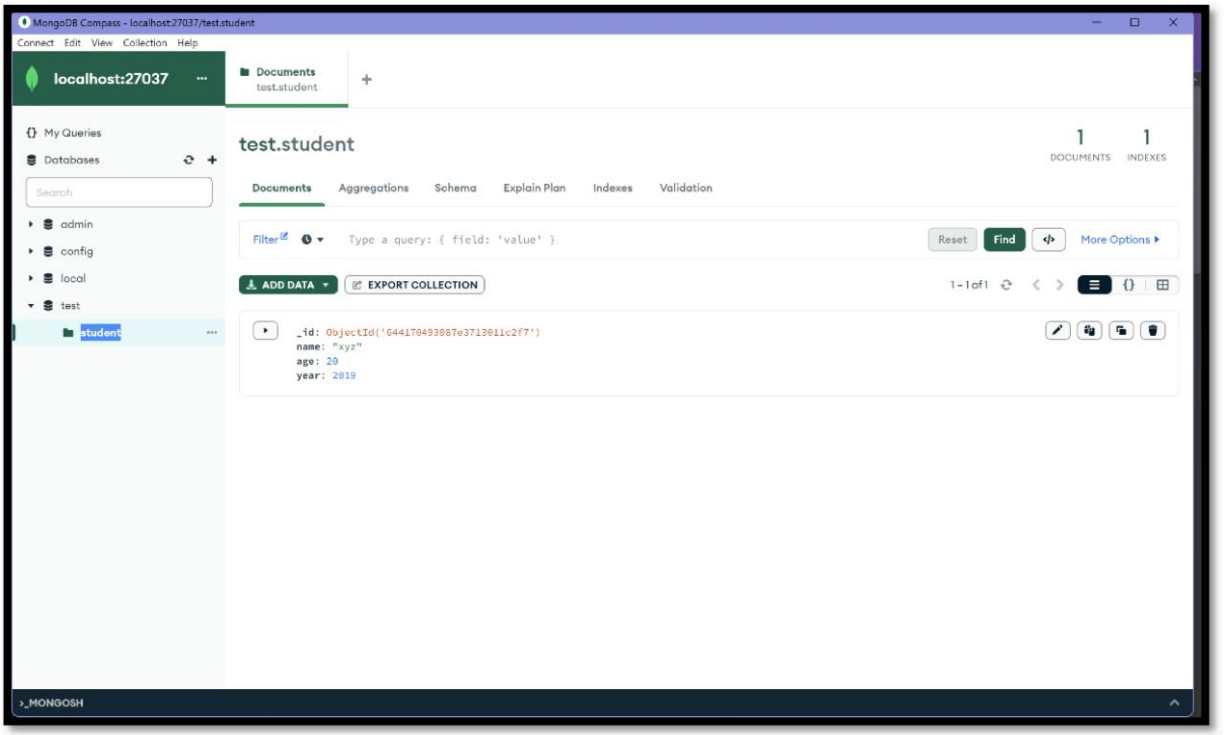
> rs.initiate()
{
"info2" : "no configuration specified. Using a default configuration for the set",
"me" : "localhost:27017",
"ok" : 1
}
rs0:SECONDARY> rs.add({ host: "127.0.0.1:27027", priority: 0, votes: 0 })
{
"ok" : 1,
"$clusterTime" : {
"clusterTime" : Timestamp(1682009843, 1),
"signature" : {
"hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
"keyId" : NumberLong(0)
}
},
"operationTime" : Timestamp(1682009843, 1)
}
rs0:PRIMARY> rs.add({ host: "127.0.0.1:27037", priority: 0, votes: 0 })
{
"ok" : 1,
"$clusterTime" : {
"clusterTime" : Timestamp(1682009851, 1),
"signature" : {
"hash" : BinData(0,"AAAAAAAAAAAAAAAAAAAAAAAAAAA="),
"keyId" : NumberLong(0)
}
},
"operationTime" : Timestamp(1682009851, 1)
}
rs0:PRIMARY> rs.status()


Connect Edit View Help
localhost:27017
My Queries
Databases
Performance
Connection info
Stats
3 DBs
17 Collections
Hosts
localhost:27017
127.0.0.1:27027
127.0.0.1:27037
Cluster
Replica Set rs0
3 Nodes
Edition
MongoDB 5.0.5 Community
Close
admin
config
local
group by responder mobile
No of users per city
update count by date
>_MONGOSH


MongoDB Compass - localhost:27017
Connect Edit View Help
localhost:27017
My Queries
Databases
Performance
Create Database
Database Name
StudentDB
Collection Name
student
Time-Series
Time-series collections efficiently store sequences of measurements over a period of time. Learn More
Additional preferences (e.g. Custom collation, Capped, Clustered collections)
Cancel
Create Database
group by responder mobile
No of users per city
update count by date
>_MONGOSH


MongoDB Compass - localhost:27017
Connect Edit View Help
localhost:27017
My Queries
Databases
Performance
Create database
Refresh
View
Sort by
Database Name
admin
Storage size: 40.96 kB
Collections: 0
Indexes: 2
config
Storage size: 61.44 kB
Collections: 5
Indexes: 11
local
Storage size: 266.24 kB
Collections: 6
Indexes: 8
>_MONGOSH
> db.student.insertOne({name:"xyz",age:20,year:2019})
{
acknowledged: true,
insertedId: ObjectId("644170493087e3713011c2f7")
}
rs0 [primary] test>





RESULT:

Hence the above program has been compiled and executed successfully.

Ex.no:11
Prg.Name: **MONGODB INDEXING**
Date:

AIM:

Determine the data cleaning loading and handling for time series.

ALGORITHM:

Step 1: Open MongoDB Compass.

Step 2: Use mongosh inside the MongoDB Compass.

Step 3: Determine the collection for which you want to create an index.

Step 4: Select the field or fields in the collection on which the index need to be created.

Step 5: Choose the appropriate index type based on the query patterns and performance requirements

STEP 6: Use the createIndex() method to create the index.

Step 7: Use find() to display the output.

Step 8: Stop the program.

SOURCE CODE:

```
db.MoviesCopy.createIndex({ Name : 1})

db.MoviesCopy.createIndex({ Hero : 1})

db.MoviesCopy.createIndex({ Director : 1})

db.MoviesCopy.createIndex({ ReleaseDate : 1})

db.MoviesCopy.getIndexes()
```

OUTPUT:

```
Hotstar> db.MoviesCopy.createIndex({ Name: 1 })
         db.MoviesCopy.createIndex({ Hero: 1 })
         db.MoviesCopy.createIndex({ Director: 1 })
         db.MoviesCopy.createIndex({ ReleaseDate: 1 })
```

```
> db.MoviesCopy.createIndex({ Name: 1 })
  db.MoviesCopy.createIndex({ Hero: 1 })
  db.MoviesCopy.createIndex({ Director: 1 })
  db.MoviesCopy.createIndex({ ReleaseDate: 1 })
< ReleaseDate_1
> db.MoviesCopy.getIndexes()
< [
    { v: 2, key: { _id: 1 }, name: '_id_' },
    { v: 2, key: { Name: 1 }, name: 'Name_1' },
    { v: 2, key: { Hero: 1 }, name: 'Hero_1' },
    { v: 2, key: { Director: 1 }, name: 'Director_1' },
    { v: 2, key: { ReleaseDate: 1 }, name: 'ReleaseDate_1' }
  ]
Hotstar>
```

RESULT:

Hence the above program has been compiled and executed successfully.