B. K. Birla College of Arts, Science & Commerce (Autonomous), Kalyan

(Department of Computer Science)

This is to certify that

Mr. **Jay Vishram Remulkar** Roll No. **40** Exam Seat No. **13228340** has satisfactorily completed the Practical in **Advanced Database Techniques** as laid down in the regulation of University of Mumbai for the purpose **Semester I** Examination **2022 - 2023**

Date: 23 January 2023

Place: Kalyan

Head

Dept. of Computer Science

Professor In-charge

Signature of Examiner Computer Science

INDEX

Practical 1

Aim: Create different types that include attributes and methods. Define tables for these types by adding a sufficient number of tuples. Demonstrate insert, update and delete operations on these tables. Execute queries on them.

Background Information:

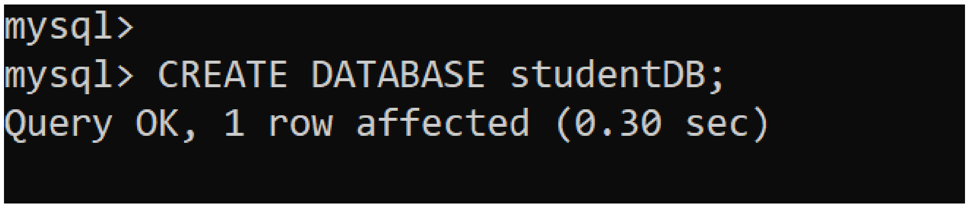
MySQL Database:

MySQL is an open-source relational database management system (RDBMS). Its name is a combination of "My", the name of co-founder Michael Widenius's daughter My, and "SQL", the acronym for Structured Query Language. A relational database organizes data into one or more data tables in which data may be related to each other; these relations help structure the data. SQL is a language programmer used to create, modify and extract data from the relational database, as well as control user access to the database. In addition to relational databases and SQL, an RDBMS like MySQL works with an operating system to implement a relational database in a computer's storage system, manages users, allows for network access and facilitates testing database integrity and creation of backups.

Code/Output:

Command to create database:

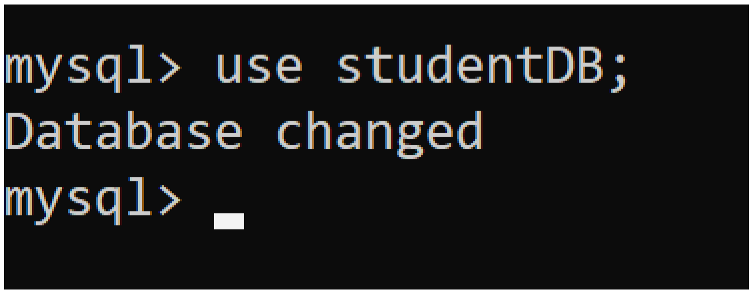
---->CREATE DATABASE studentDB;



```
mysql>
mysql> CREATE DATABASE studentDB;
Query OK, 1 row affected (0.30 sec)
```

Command to change database:

----> use studentDB;



```
mysql> use studentDB;
Database changed
mysql>
```

Command to create a table:

---->CREATE TABLE studentData(

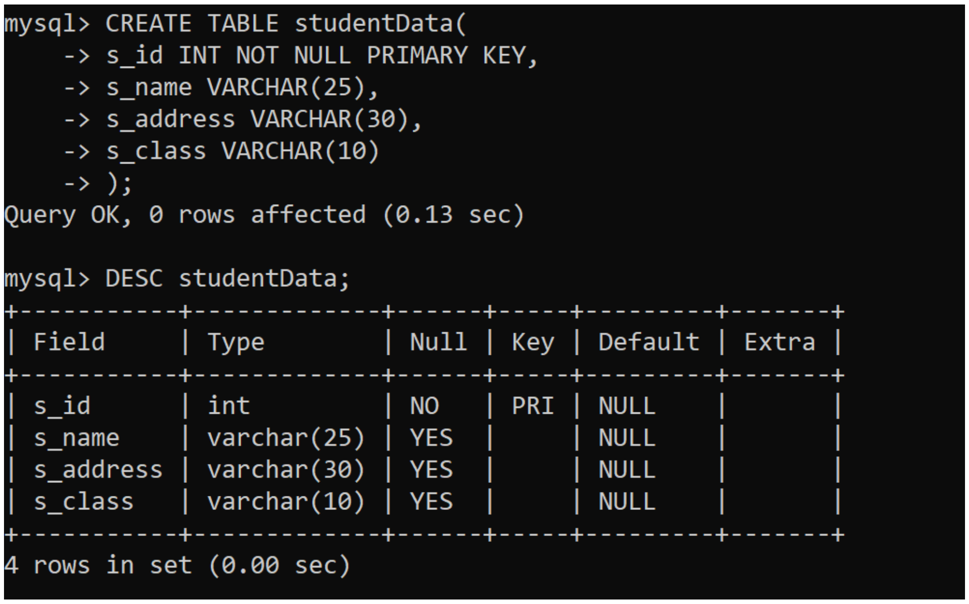
s_id INT NOT NULL PRIMARY KEY,

s_name VARCHAR(25),

s_address VARCHAR(30),

s_class VARCHAR(10)

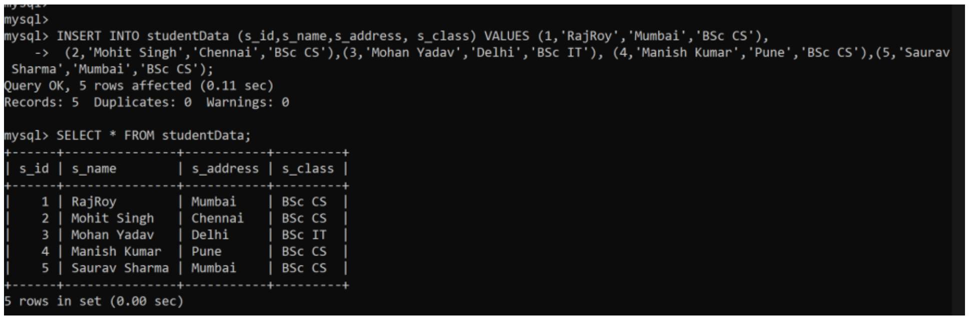
);



```
mysql> CREATE TABLE studentData(
    -> s_id INT NOT NULL PRIMARY KEY,
    -> s_name VARCHAR(25),
    -> s_address VARCHAR(30),
    -> s_class VARCHAR(10)
    -> );
Query OK, 0 rows affected (0.13 sec)

mysql> DESC studentData;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| s_id      | int         | NO   | PRI | NULL    |       |
| s_name    | varchar(25) | YES  |     | NULL    |       |
| s_address | varchar(30) | YES  |     | NULL    |       |
| s_class   | varchar(10) | YES  |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

Command to insert rows:

---->INSERT INTO studentData (s_id,s_name,s_address, s_class) VALUES (1,'RajRoy','Mumbai','BSc CS'),(2,'Mohit Singh','Chennai','BSc CS'),(3,'Mohan Yadav','Delhi','BSc IT'), (4,'Manish Kumar','Pune','BSc CS'),(5,'Saurav Sharma','Mumbai','BSc CS');



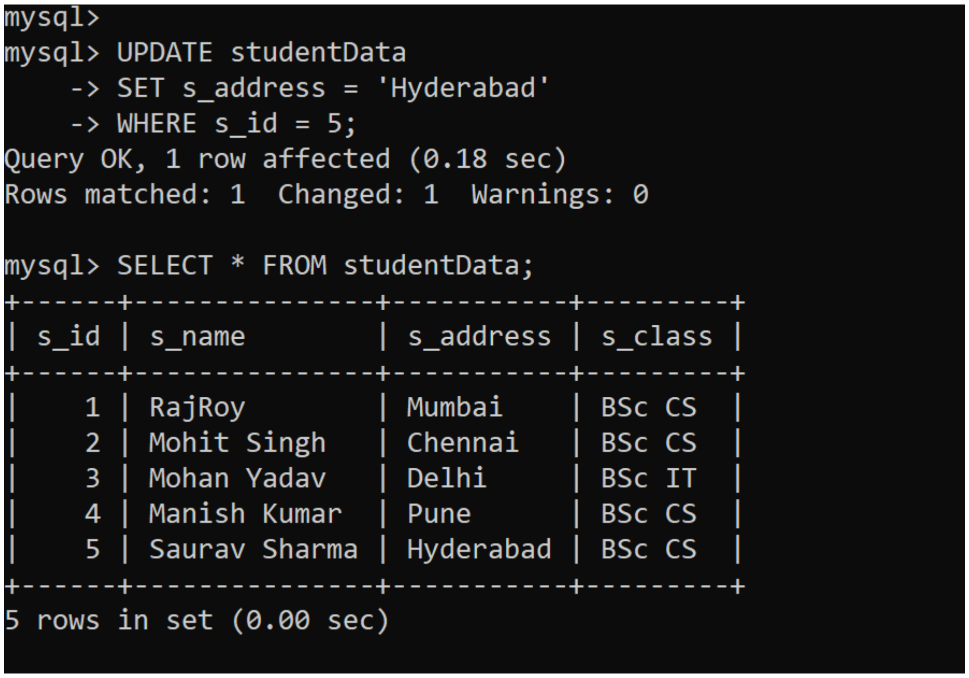
```
mysql>
mysql> INSERT INTO studentData (s_id,s_name,s_address, s_class) VALUES (1,'RajRoy','Mumbai','BSc CS'),
    ->  (2,'Mohit Singh','Chennai','BSc CS'),(3,'Mohan Yadav','Delhi','BSc IT'), (4,'Manish Kumar','Pune','BSc CS'),(5,'Saurav
 Sharma','Mumbai','BSc CS');
Query OK, 5 rows affected (0.11 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM studentData;
+------+---------------+-----------+---------+
| s_id | s_name        | s_address | s_class |
+------+---------------+-----------+---------+
|    1 | RajRoy        | Mumbai    | BSc CS  |
|    2 | Mohit Singh   | Chennai   | BSc CS  |
|    3 | Mohan Yadav   | Delhi     | BSc IT  |
|    4 | Manish Kumar  | Pune      | BSc CS  |
|    5 | Saurav Sharma | Mumbai    | BSc CS  |
+------+---------------+-----------+---------+
5 rows in set (0.00 sec)
```

Command to update a data:

UPDATE studentData

SET s_address = 'Hyderabad'

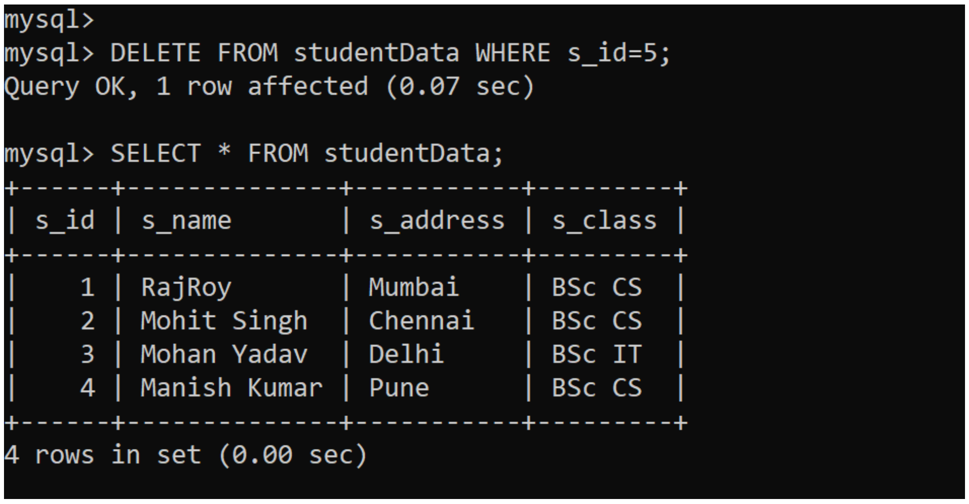
WHERE s_id = 5;



```
mysql>
mysql> UPDATE studentData
    -> SET s_address = 'Hyderabad'
    -> WHERE s_id = 5;
Query OK, 1 row affected (0.18 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM studentData;
+------+--------------+-----------+---------+
| s_id | s_name       | s_address | s_class |
+------+--------------+-----------+---------+
|    1 | RajRoy       | Mumbai    | BSc CS  |
|    2 | Mohit Singh  | Chennai   | BSc CS  |
|    3 | Mohan Yadav  | Delhi     | BSc IT  |
|    4 | Manish Kumar | Pune      | BSc CS  |
|    5 | Saurav Sharma | Hyderabad | BSc CS  |
+------+--------------+-----------+---------+
5 rows in set (0.00 sec)
```

Command to delete a data:

---->DELETE FROM studentData WHERE s_id=5;



```
mysql>
mysql> DELETE FROM studentData WHERE s_id=5;
Query OK, 1 row affected (0.07 sec)

mysql> SELECT * FROM studentData;
+------+--------------+-----------+---------+
| s_id | s_name       | s_address | s_class |
+------+--------------+-----------+---------+
|    1 | RajRoy       | Mumbai    | BSc CS  |
|    2 | Mohit Singh  | Chennai   | BSc CS  |
|    3 | Mohan Yadav  | Delhi     | BSc IT  |
|    4 | Manish Kumar | Pune      | BSc CS  |
+------+--------------+-----------+---------+
4 rows in set (0.00 sec)
```

Practical 4

Aim: Create a table that stores spatial data and issues queries on it.

Background Information:

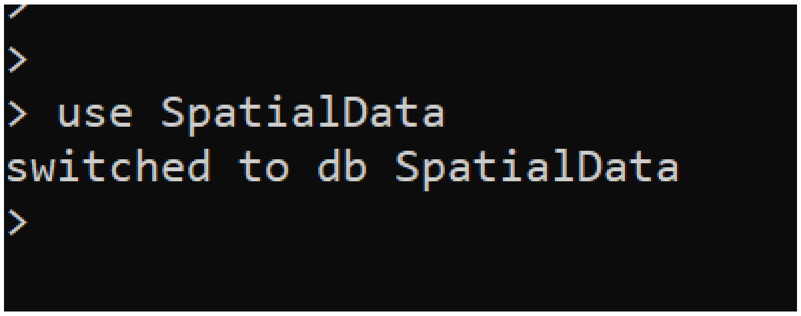
Spatial Database:

A spatial database is a general-purpose database (usually a relational database) that has been enhanced to include spatial data that represents objects defined in a geometric space, along with tools for querying and analyzing such data. Most spatial databases allow the representation of simple geometric objects such as points, lines and polygons. Some spatial databases handle more complex structures such as 3D objects, topological coverages, linear networks, and triangulated irregular networks (TINs). While typical databases have been developed to manage various numeric and character types of data, such databases require additional functionality to process spatial data types efficiently, and developers have often added geometry or feature data types. The Open Geospatial Consortium (OGC) developed the Simple Features specification (first released in 1997)[1] and sets standards for adding spatial functionality to database systems.[2] The SQL/MM Spatial ISO/IEC standard is a part the SQL/MM multimedia standard and extends the Simple Features standard with data types that support circular interpolations.[3] Almost all current relational and object-relational database management systems now have spatial extensions, and some GIS software vendors have developed their own spatial extensions to database management systems.

Code/Output:

Command to create database:

---->use SpatialData



```
>
> use SpatialData
switched to db SpatialData
>
```

Command to insert data:

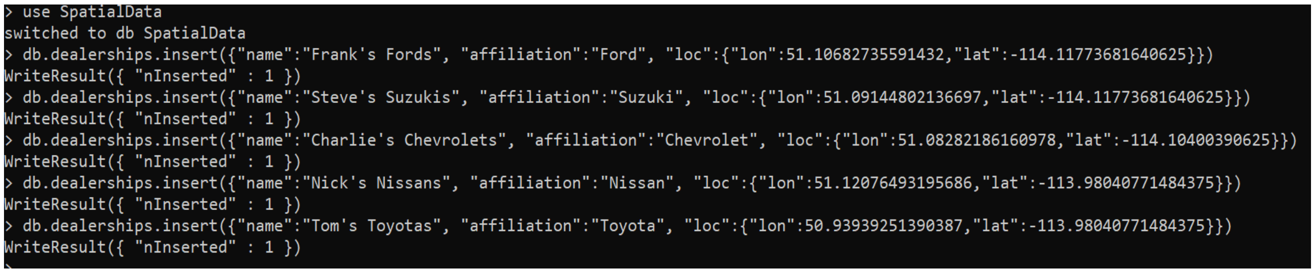
---->db.dealerships.insert({"name":"Frank's Fords", "affiliation":"Ford", "loc":{"lon":51.10682735591432,"lat":-114.11773681640625}})

---->db.dealerships.insert({"name":"Steve's Suzukis", "affiliation":"Suzuki", "loc":{"lon":51.09144802136697,"lat":-114.11773681640625}})

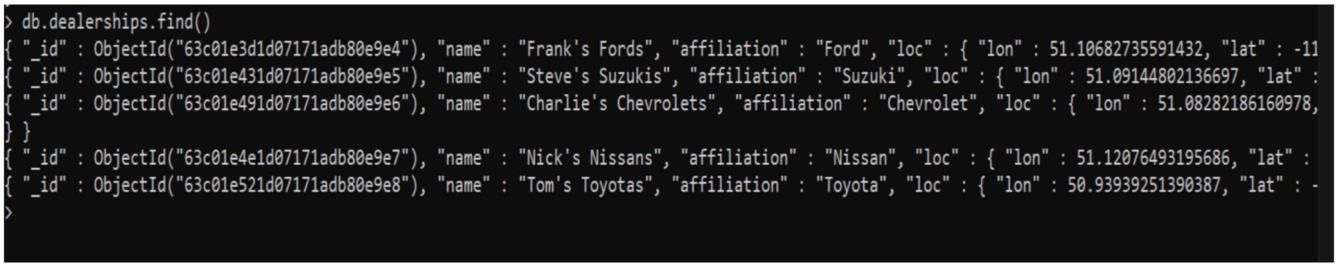
---->db.dealerships.insert({"name":"Charlie's Chevrolets", "affiliation":"Chevrolet", "loc":{"lon":51.08282186160978,"lat":-114.10400390625}})

---->db.dealerships.insert({"name":"Nick's Nissans", "affiliation":"Nissan", "loc":{"lon":51.12076493195686,"lat":-113.98040771484375}})

---->db.dealerships.insert({"name":"Tom's Toyotas", "affiliation":"Toyota", "loc":{"lon":50.93939251390387,"lat":-113.98040771484375}})



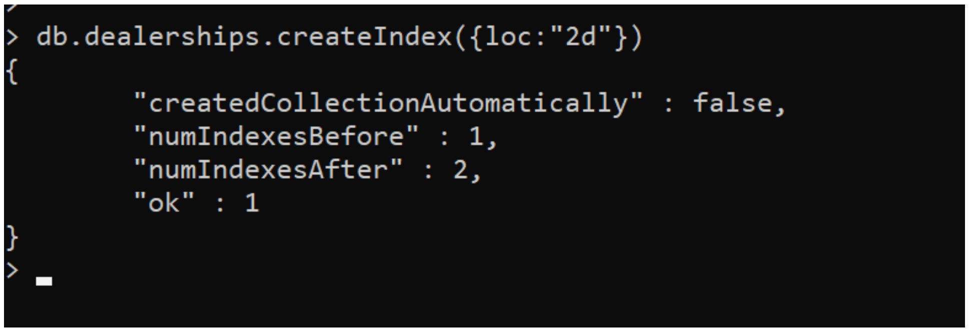
```
> use SpatialData
switched to db SpatialData
> db.dealerships.insert({"name":"Frank's Fords", "affiliation":"Ford", "loc":{"lon":51.10682735591432,"lat":-114.11773681640625}})
WriteResult({ "nInserted" : 1 })
> db.dealerships.insert({"name":"Steve's Suzukis", "affiliation":"Suzuki", "loc":{"lon":51.09144802136697,"lat":-114.11773681640625}})
WriteResult({ "nInserted" : 1 })
> db.dealerships.insert({"name":"Charlie's Chevrolets", "affiliation":"Chevrolet", "loc":{"lon":51.08282186160978,"lat":-114.10400390625}})
WriteResult({ "nInserted" : 1 })
> db.dealerships.insert({"name":"Nick's Nissans", "affiliation":"Nissan", "loc":{"lon":51.12076493195686,"lat":-113.98040771484375}})
WriteResult({ "nInserted" : 1 })
> db.dealerships.insert({"name":"Tom's Toyotas", "affiliation":"Toyota", "loc":{"lon":50.93939251390387,"lat":-113.98040771484375}})
WriteResult({ "nInserted" : 1 })
```



```
> db.dealerships.find()
{ "_id" : ObjectId("63c01e3d1d07171adb80e9e4"), "name" : "Frank's Fords", "affiliation" : "Ford", "loc" : { "lon" : 51.10682735591432, "lat" : -11
{ "_id" : ObjectId("63c01e431d07171adb80e9e5"), "name" : "Steve's Suzukis", "affiliation" : "Suzuki", "loc" : { "lon" : 51.09144802136697, "lat" :
{ "_id" : ObjectId("63c01e491d07171adb80e9e6"), "name" : "Charlie's Chevrolets", "affiliation" : "Chevrolet", "loc" : { "lon" : 51.08282186160978,
} }
{ "_id" : ObjectId("63c01e4e1d07171adb80e9e7"), "name" : "Nick's Nissans", "affiliation" : "Nissan", "loc" : { "lon" : 51.12076493195686, "lat" :
{ "_id" : ObjectId("63c01e521d07171adb80e9e8"), "name" : "Tom's Toyotas", "affiliation" : "Toyota", "loc" : { "lon" : 50.93939251390387, "lat" : -
>
```

Command to create index on loc field:

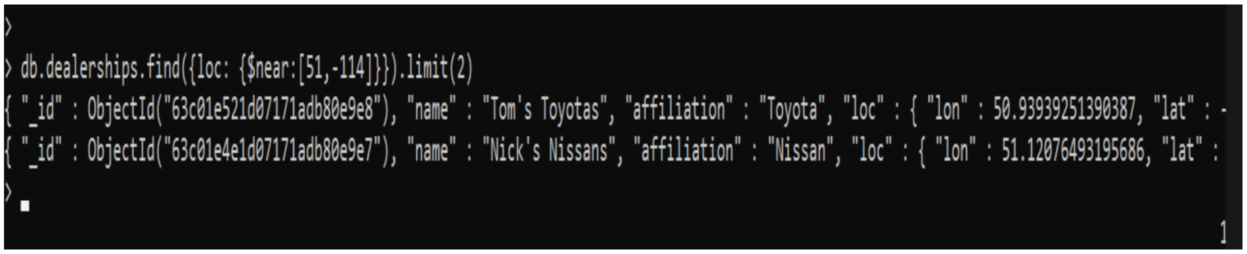
---->db.dealerships.createIndex({loc:"2d"})



```
> db.dealerships.createIndex({loc:"2d"})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
>
```

Command to get two dealerships closest to a specific co-ordinate:

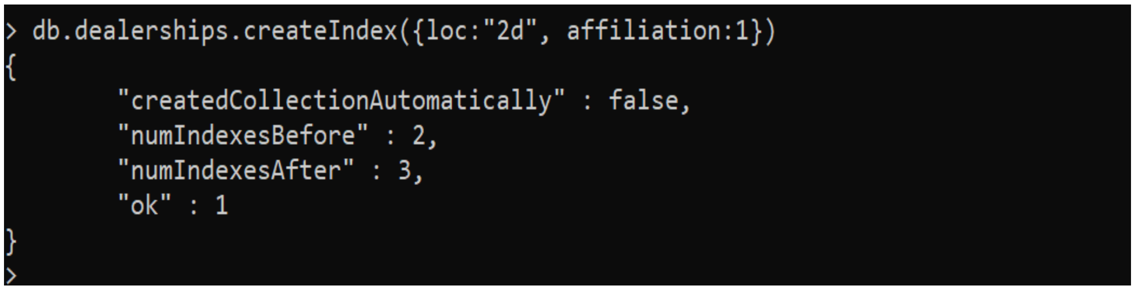
---->db.dealerships.find({loc: {$near:[51,-114]}}).limit(2)



```
>
> db.dealerships.find({loc: {$near:[51,-114]}}).limit(2)
{ "_id" : ObjectId("63c01e521d07171adb80e9e8"), "name" : "Tom's Toyotas", "affiliation" : "Toyota", "loc" : { "lon" : 50.93939251390387, "lat" : -
{ "_id" : ObjectId("63c01e4e1d07171adb80e9e7"), "name" : "Nick's Nissans", "affiliation" : "Nissan", "loc" : { "lon" : 51.12076493195686, "lat" :
>
```

Command to create compound index (On multiple fields):

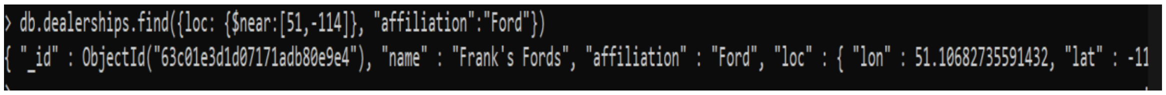
---->db.dealerships.createIndex({loc:"2d", affiliation:1})



```
> db.dealerships.createIndex({loc:"2d", affiliation:1})
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 2,
        "numIndexesAfter" : 3,
        "ok" : 1
}
>
```

Command to filter data by dealership affiliation:

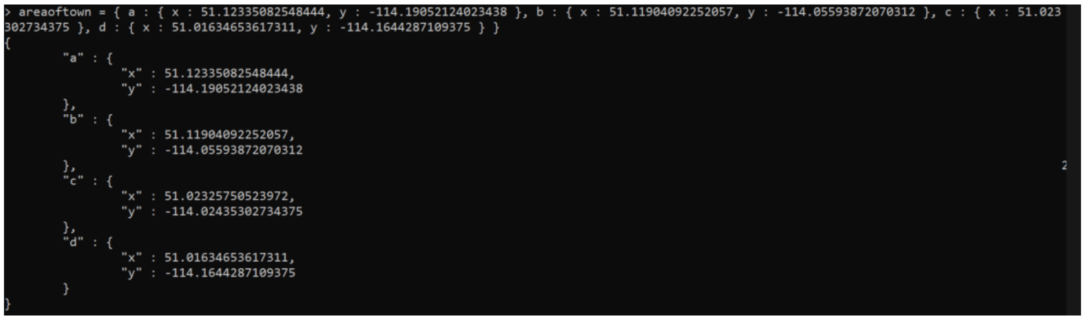
---->db.dealerships.find({loc: {$near:[51,-114]}, "affiliation":"Ford"})



```
> db.dealerships.find({loc: {$near:[51,-114]}, "affiliation":"Ford"})
{ "_id" : ObjectId("63c01e3d1d07171adb80e9e4"), "name" : "Frank's Fords", "affiliation" : "Ford", "loc" : { "lon" : 51.10682735591432, "lat" : -11
```

Command to define a polygon for a specific area of town:

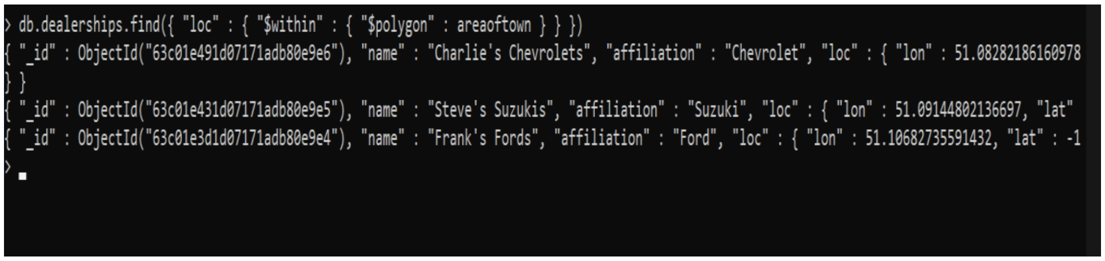
---->areaoftown = { a : { x : 51.12335082548444, y : -114.19052124023438 }, b : { x : 51.11904092252057, y : -114.05593872070312 }, c : { x : 51.02325750523972, y : -114.02435302734375 }, d : { x : 51.01634653617311, y : -114.1644287109375 } }



```
> areaoftown = { a : { x : 51.12335082548444, y : -114.19052124023438 }, b : { x : 51.11904092252057, y : -114.05593872070312 }, c : { x : 51.023
302734375 }, d : { x : 51.01634653617311, y : -114.1644287109375 } }
{
        "a" : {
                "x" : 51.12335082548444,
                "y" : -114.19052124023438
        },
        "b" : {
                "x" : 51.11904092252057,
                "y" : -114.05593872070312
        },
        "c" : {
                "x" : 51.02325750523972,
                "y" : -114.02435302734375
        },
        "d" : {
                "x" : 51.01634653617311,
                "y" : -114.1644287109375
        }
}
```

Command to search for all dealerships within a given area of town:

---->db.dealerships.find({ "loc" : { "$within" : { "$polygon" : areaoftown } } })



```
> db.dealerships.find({ "loc" : { "$within" : { "$polygon" : areaoftown } } })
{ "_id" : ObjectId("63c01e491d07171adb80e9e6"), "name" : "Charlie's Chevrolets", "affiliation" : "Chevrolet", "loc" : { "lon" : 51.0828218616097
} }
{ "_id" : ObjectId("63c01e431d07171adb80e9e5"), "name" : "Steve's Suzukis", "affiliation" : "Suzuki", "loc" : { "lon" : 51.09144802136697, "lat"
{ "_id" : ObjectId("63c01e3d1d07171adb80e9e4"), "name" : "Frank's Fords", "affiliation" : "Ford", "loc" : { "lon" : 51.10682735591432, "lat" : -1
> ■
```

Practical 5

Aim: Create a table that stores spatial data and issues queries on it.

Background Information:

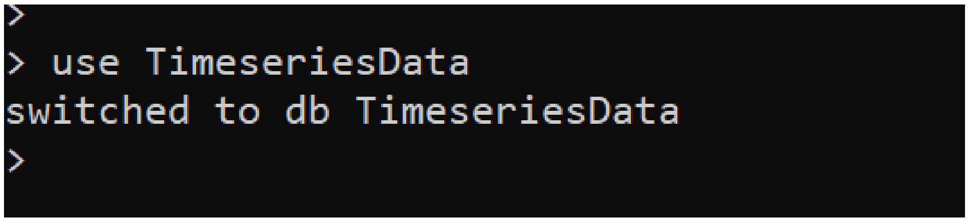
Temporal Database:

Temporal Database is a database with built-in support for handling time sensitive data. Usually, databases store information only about current state, and not about past states. For example, in a employee database if the address or salary of a particular person changes, the database gets updated, the old value is no longer there. However, for many applications, it is important to maintain the past or historical values and the time at which the data was updated. That is, knowledge of evolution is required. That is where temporal databases are useful. It stores information about the past, present and future. Any data that is time dependent is called temporal data and these are stored in temporal databases.

Code/Output:

Command to create database:

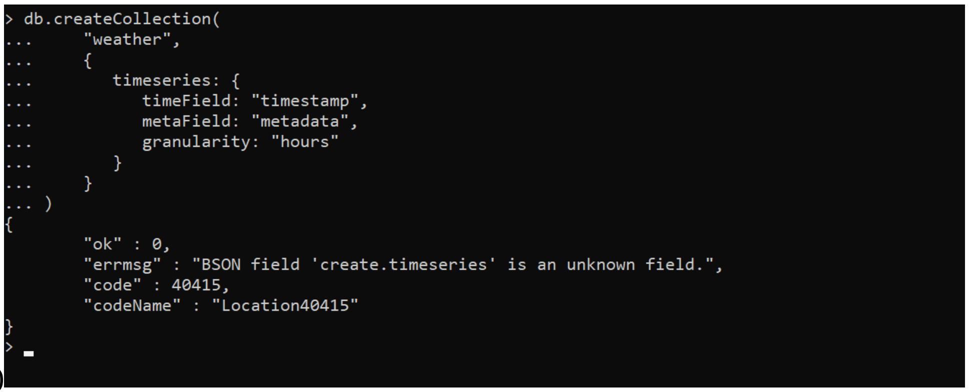
---->use TimeseriesData



```
>
> use TimeseriesData
switched to db TimeseriesData
>
```

Command to make a collection:

```
---->db.createCollection(
    "weather",
    {
       timeseries: {
          timeField: "timestamp",
          metaField: "metadata",
          granularity: "hours"
       }
    }
```

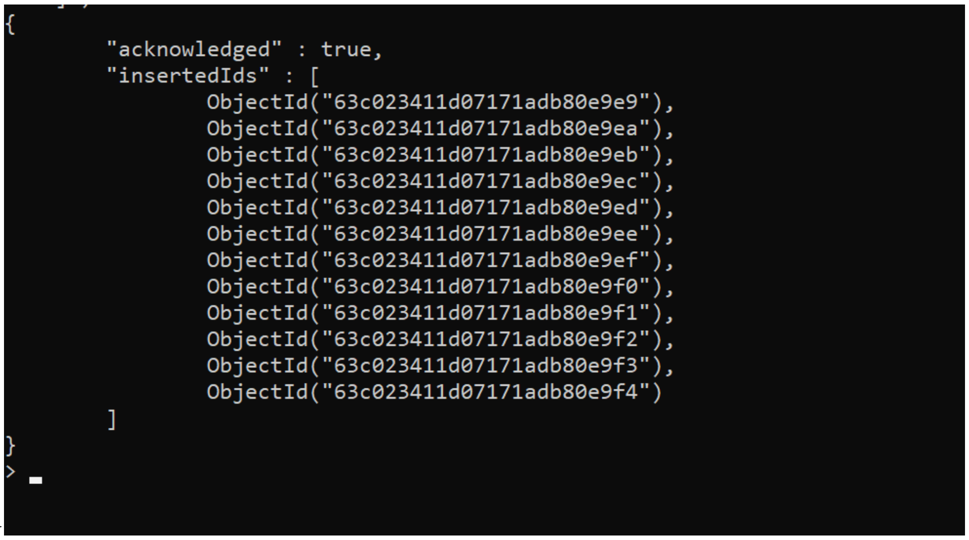


```
> db.createCollection(
...     "weather",
...     {
...        timeseries: {
...           timeField: "timestamp",
...           metaField: "metadata",
...           granularity: "hours"
...        }
...     }
... )
{
        "ok" : 0,
        "errmsg" : "BSON field 'create.timeseries' is an unknown field.",
        "code" : 40415,
        "codeName" : "Location40415"
}
> 
```

)

Command to insert multiple data:

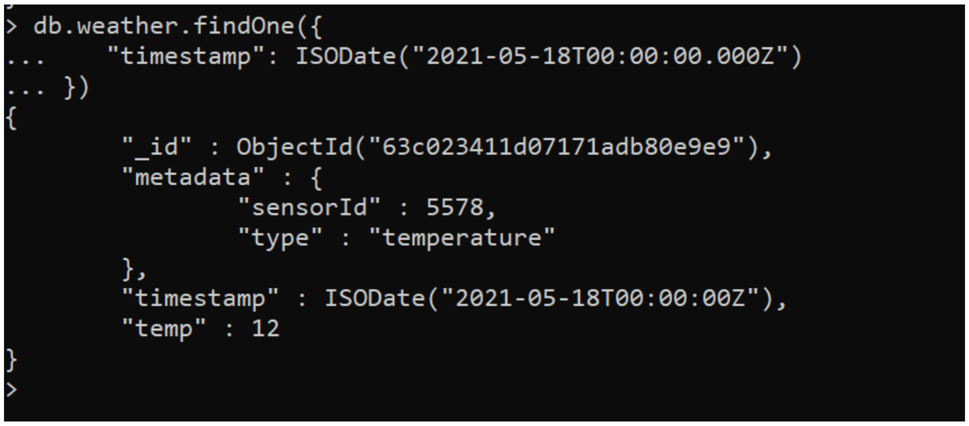
```
---->db.weather.insertMany( [
   {
      "metadata": { "sensorId": 5578, "type": "temperature" },
      "timestamp": ISODate("2021-05-18T00:00:00.000Z"),
      "temp": 12
   },
   {
      "metadata": { "sensorId": 5578, "type": "temperature" },
      "timestamp": ISODate("2021-05-18T04:00:00.000Z"),
      "temp": 11
   },
   {
      "metadata": { "sensorId": 5578, "type": "temperature" },
      "timestamp": ISODate("2021-05-18T08:00:00.000Z"),
      "temp": 11
   },
   {
      "metadata": { "sensorId": 5578, "type": "temperature" },
      "timestamp": ISODate("2021-05-18T12:00:00.000Z"),
      "temp": 12
   },
   {
      "metadata": { "sensorId": 5578, "type": "temperature" },
      "timestamp": ISODate("2021-05-18T16:00:00.000Z"),
      "temp": 16
   },
   {
      "metadata": { "sensorId": 5578, "type": "temperature" },
      "timestamp": ISODate("2021-05-18T20:00:00.000Z"),
      "temp": 15
   }, {
      "metadata": { "sensorId": 5578, "type": "temperature" },
      "timestamp": ISODate("2021-05-19T00:00:00.000Z"),
      "temp": 13
   },
   {
      "metadata": { "sensorId": 5578, "type": "temperature" },
      "timestamp": ISODate("2021-05-19T04:00:00.000Z"),
      "temp": 12
   },
   {
      "metadata": { "sensorId": 5578, "type": "temperature" },
      "timestamp": ISODate("2021-05-19T08:00:00.000Z"),
      "temp": 11
   },
   {
      "metadata": { "sensorId": 5578, "type": "temperature" },
      "timestamp": ISODate("2021-05-19T12:00:00.000Z"),
      "temp": 12
   },
   {
      "metadata": { "sensorId": 5578, "type": "temperature" },
      "timestamp": ISODate("2021-05-19T16:00:00.000Z"),
      "temp": 17
   },
   {
      "metadata": { "sensorId": 5578, "type": "temperature" },
      "timestamp": ISODate("2021-05-19T20:00:00.000Z"),
      "temp": 12
```


```
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("63c023411d07171adb80e9e9"),
                ObjectId("63c023411d07171adb80e9ea"),
                ObjectId("63c023411d07171adb80e9eb"),
                ObjectId("63c023411d07171adb80e9ec"),
                ObjectId("63c023411d07171adb80e9ed"),
                ObjectId("63c023411d07171adb80e9ee"),
                ObjectId("63c023411d07171adb80e9ef"),
                ObjectId("63c023411d07171adb80e9f0"),
                ObjectId("63c023411d07171adb80e9f1"),
                ObjectId("63c023411d07171adb80e9f2"),
                ObjectId("63c023411d07171adb80e9f3"),
                ObjectId("63c023411d07171adb80e9f4")
        ]
}
>
```

}

Command to view a single data:

```
---->db.weather.findOne({

   "timestamp": ISODate("2021-05-18T00:00:00.000Z")

})
```


```
> db.weather.findOne({
...     "timestamp": ISODate("2021-05-18T00:00:00.000Z")
... })
{
        "_id" : ObjectId("63c023411d07171adb80e9e9"),
        "metadata" : {
                "sensorId" : 5578,
                "type" : "temperature"
        },
        "timestamp" : ISODate("2021-05-18T00:00:00Z"),
        "temp" : 12
}
>
```

Practical 6

Aim: Demonstrate the Accessing and Storing and performing CRUD operations in

1. MongoDB
2. Redis

Background Information:

MongoDB:

MongoDB is a source-available cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with optional schemas. MongoDB is developed by MongoDB Inc. and licensed under the Server-Side Public License (SSPL) which is deemed non-free by several distributions.

Redis:

Redis is an in-memory data structure store, used as a distributed, in-memory key–value database, cache and message broker, with optional durability. Redis supports different kinds of abstract data structures, such as strings, lists, maps, sets, sorted sets, Hyper Loglogs, bitmaps, streams, and spatial indices. The project was developed and maintained by Salvatore Sanfilippo, starting in 2009. From 2015 until 2020, he led a project core team sponsored by Redis Labs. Salvatore Sanfilippo left Redis as the maintainer in 2020. It is open-source software released under a BSD 3-clause license. In 2021, not long after the original author and main maintainer left, Redis Labs dropped the Labs from its name and now is known simply as "Redis".

Create Operation:

The create or insert operations are used to insert or add new documents in the collection. If a collection does not exist, then it will create a new collection in the database.

Read Operation:

The Read operations are used to retrieve documents from the collection, or in other words, read operations are used to query a collection for a document.

Update Operation:

The update operations are used to update or modify the existing document in the collection.

Delete Operation:

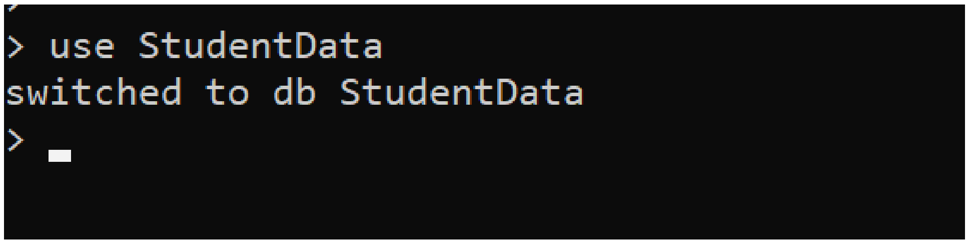
The delete operation is used to delete or remove documents from a collection.

Code/Output:

1. MongoDB

Command to create a Database

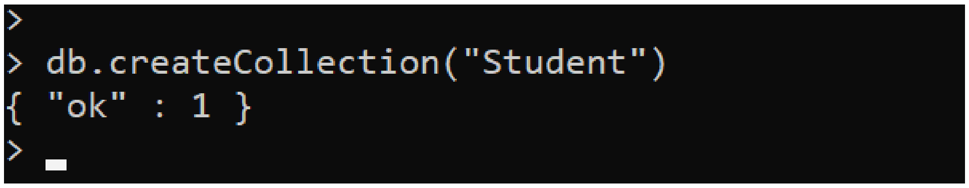
---->use StudentData



```
> use StudentData
switched to db StudentData
> 
```

Command to create a Collection

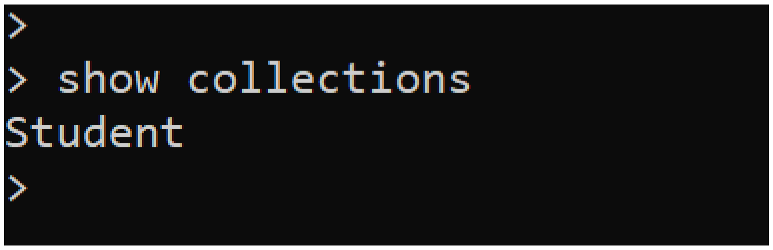
---->db.createCollection("Student")



```
>
> db.createCollection("Student")
{ "ok" : 1 }
>
```

Command to view Collection

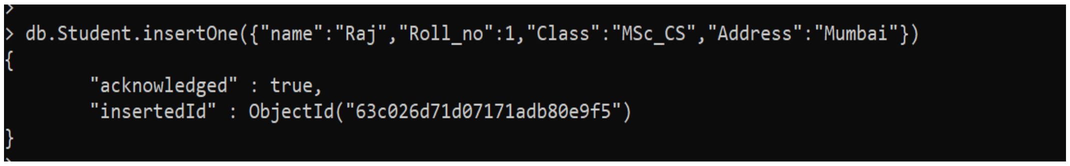
---->show collections



```
>
> show collections
Student
>
```

Command to insert a single document in Collection

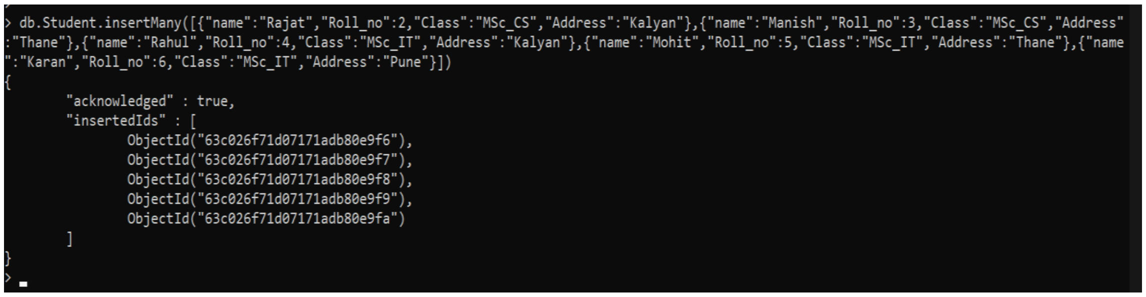
---->db.Student.insertOne({"name":"Raj","Roll_no":1,"Class":"MSc_CS","Address":"Mumbai"})



```
> db.Student.insertOne({"name":"Raj","Roll_no":1,"Class":"MSc_CS","Address":"Mumbai"})
{
        "acknowledged" : true,
        "insertedId" : ObjectId("63c026d71d07171adb80e9f5")
}
```

Command to insert multiple documents in Collection

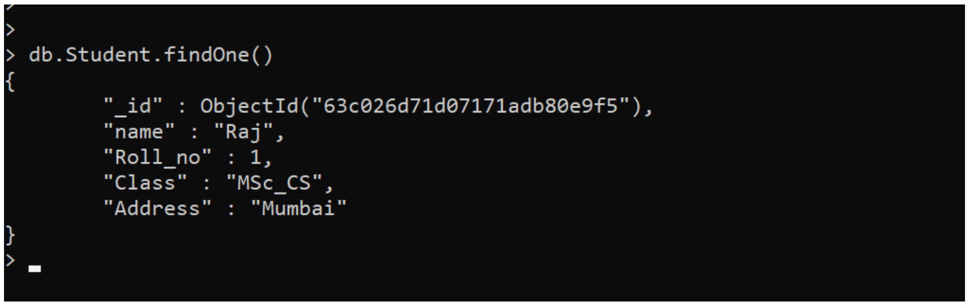
>db.Student.insertMany([{"name":"Rajat","Roll_no":2,"Class":"MSc_CS","Address":"Kalyan"},{"name":"Manish","Roll_no":3,"Class":"MSc_CS","Address":"Thane"},{"name":"Rahul","Roll_no":4,"Class":"MSc_IT","Address":"Kalyan"},{"name":"Mohit","Roll_no":5,"Class":"MSc_IT","Address":"Thane"},{"name":"Karan","Roll_no":6,"Class":"MSc_IT","Address":"Pune"}])



```
> db.Student.insertMany([{"name":"Rajat","Roll_no":2,"Class":"MSc_CS","Address":"Kalyan"},{"name":"Manish","Roll_no":3,"Class":"MSc_CS","Address"
:"Thane"},{"name":"Rahul","Roll_no":4,"Class":"MSc_IT","Address":"Kalyan"},{"name":"Mohit","Roll_no":5,"Class":"MSc_IT","Address":"Thane"},{"name
":"Karan","Roll_no":6,"Class":"MSc_IT","Address":"Pune"}])
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("63c026f71d07171adb80e9f6"),
                ObjectId("63c026f71d07171adb80e9f7"),
                ObjectId("63c026f71d07171adb80e9f8"),
                ObjectId("63c026f71d07171adb80e9f9"),
                ObjectId("63c026f71d07171adb80e9fa")
        ]
}
>
```

Command to view a single document(First document)

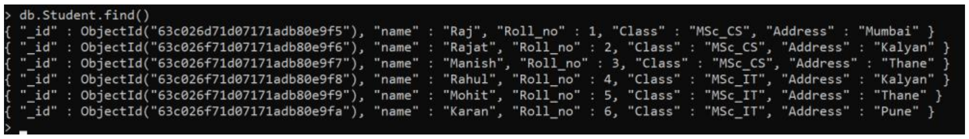
---->db.Student.findOne()



```
>
> db.Student.findOne()
{
        "_id" : ObjectId("63c026d71d07171adb80e9f5"),
        "name" : "Raj",
        "Roll_no" : 1,
        "Class" : "MSc_CS",
        "Address" : "Mumbai"
}
> 
```

Command to view all documents

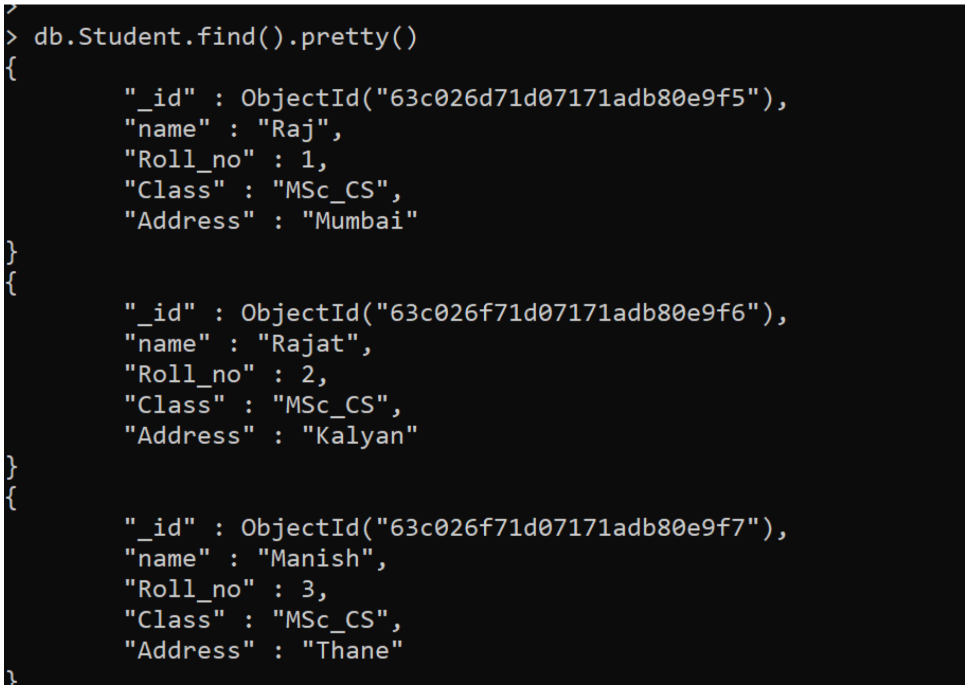
---->db.Student.find()



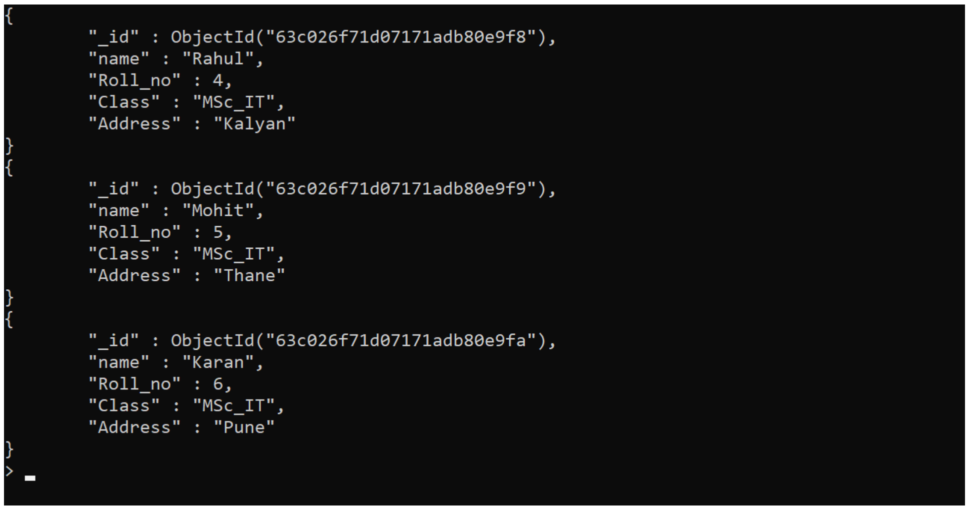
```
> db.Student.find()
{ "_id" : ObjectId("63c026d71d07171adb80e9f5"), "name" : "Raj", "Roll_no" : 1, "Class" : "MSc_CS", "Address" : "Mumbai" }
{ "_id" : ObjectId("63c026f71d07171adb80e9f6"), "name" : "Rajat", "Roll_no" : 2, "Class" : "MSc_CS", "Address" : "Kalyan" }
{ "_id" : ObjectId("63c026f71d07171adb80e9f7"), "name" : "Manish", "Roll_no" : 3, "Class" : "MSc_CS", "Address" : "Thane" }
{ "_id" : ObjectId("63c026f71d07171adb80e9f8"), "name" : "Rahul", "Roll_no" : 4, "Class" : "MSc_IT", "Address" : "Kalyan" }
{ "_id" : ObjectId("63c026f71d07171adb80e9f9"), "name" : "Mohit", "Roll_no" : 5, "Class" : "MSc_IT", "Address" : "Thane" }
{ "_id" : ObjectId("63c026f71d07171adb80e9fa"), "name" : "Karan", "Roll_no" : 6, "Class" : "MSc_IT", "Address" : "Pune" }
> 
```

Command to view documents with pretty()

---->db.Student.find().pretty()



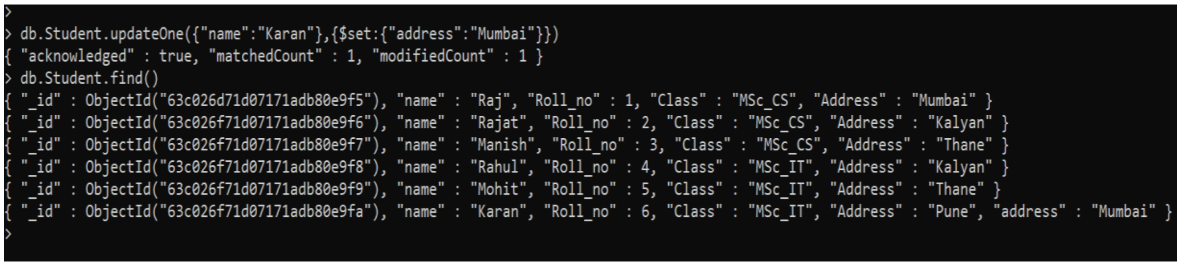
```
>
> db.Student.find().pretty()
{
        "_id" : ObjectId("63c026d71d07171adb80e9f5"),
        "name" : "Raj",
        "Roll_no" : 1,
        "Class" : "MSc_CS",
        "Address" : "Mumbai"
}
{
        "_id" : ObjectId("63c026f71d07171adb80e9f6"),
        "name" : "Rajat",
        "Roll_no" : 2,
        "Class" : "MSc_CS",
        "Address" : "Kalyan"
}
{
        "_id" : ObjectId("63c026f71d07171adb80e9f7"),
        "name" : "Manish",
        "Roll_no" : 3,
        "Class" : "MSc_CS",
        "Address" : "Thane"
}
```



```
{
        "_id" : ObjectId("63c026f71d07171adb80e9f8"),
        "name" : "Rahul",
        "Roll_no" : 4,
        "Class" : "MSc_IT",
        "Address" : "Kalyan"
}
{
        "_id" : ObjectId("63c026f71d07171adb80e9f9"),
        "name" : "Mohit",
        "Roll_no" : 5,
        "Class" : "MSc_IT",
        "Address" : "Thane"
}
{
        "_id" : ObjectId("63c026f71d07171adb80e9fa"),
        "name" : "Karan",
        "Roll_no" : 6,
        "Class" : "MSc_IT",
        "Address" : "Pune"
}
>
```

Command to update a single document

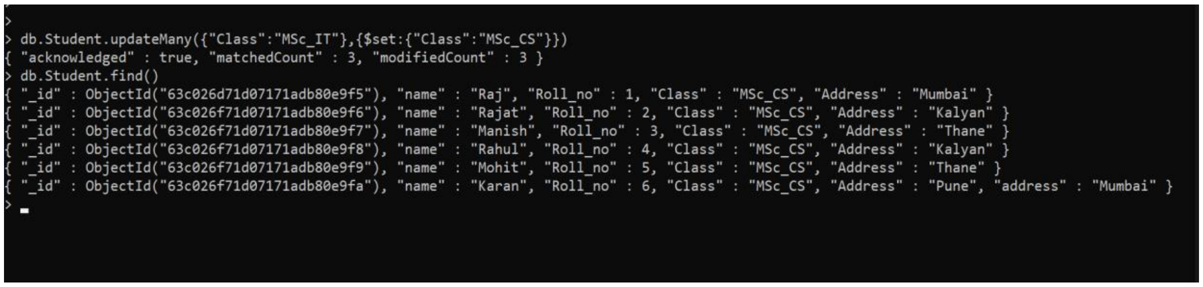
---->db.Student.updateOne({"name":"Karan"},{$set:{"address":"Mumbai"}})



```
>
> db.Student.updateOne({"name":"Karan"},{$set:{"address":"Mumbai"}})
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
> db.Student.find()
{ "_id" : ObjectId("63c026d71d07171adb80e9f5"), "name" : "Raj", "Roll_no" : 1, "Class" : "MSc_CS", "Address" : "Mumbai" }
{ "_id" : ObjectId("63c026f71d07171adb80e9f6"), "name" : "Rajat", "Roll_no" : 2, "Class" : "MSc_CS", "Address" : "Kalyan" }
{ "_id" : ObjectId("63c026f71d07171adb80e9f7"), "name" : "Manish", "Roll_no" : 3, "Class" : "MSc_CS", "Address" : "Thane" }
{ "_id" : ObjectId("63c026f71d07171adb80e9f8"), "name" : "Rahul", "Roll_no" : 4, "Class" : "MSc_IT", "Address" : "Kalyan" }
{ "_id" : ObjectId("63c026f71d07171adb80e9f9"), "name" : "Mohit", "Roll_no" : 5, "Class" : "MSc_IT", "Address" : "Thane" }
{ "_id" : ObjectId("63c026f71d07171adb80e9fa"), "name" : "Karan", "Roll_no" : 6, "Class" : "MSc_IT", "Address" : "Pune", "address" : "Mumbai" }
>
```

Command to update many documents which satisfies the conditon

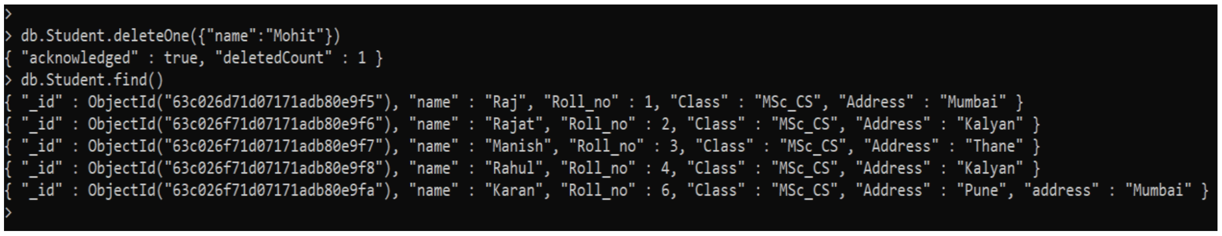
---->db.Student.updateMany({"Class":"MSc_IT"},{$set:{"Class":"MSc_CS"}})



```
>
> db.Student.updateMany({"Class":"MSc_IT"},{$set:{"Class":"MSc_CS"}})
{ "acknowledged" : true, "matchedCount" : 3, "modifiedCount" : 3 }
> db.Student.find()
{ "_id" : ObjectId("63c026d71d07171adb80e9f5"), "name" : "Raj", "Roll_no" : 1, "Class" : "MSc_CS", "Address" : "Mumbai" }
{ "_id" : ObjectId("63c026f71d07171adb80e9f6"), "name" : "Rajat", "Roll_no" : 2, "Class" : "MSc_CS", "Address" : "Kalyan" }
{ "_id" : ObjectId("63c026f71d07171adb80e9f7"), "name" : "Manish", "Roll_no" : 3, "Class" : "MSc_CS", "Address" : "Thane" }
{ "_id" : ObjectId("63c026f71d07171adb80e9f8"), "name" : "Rahul", "Roll_no" : 4, "Class" : "MSc_CS", "Address" : "Kalyan" }
{ "_id" : ObjectId("63c026f71d07171adb80e9f9"), "name" : "Mohit", "Roll_no" : 5, "Class" : "MSc_CS", "Address" : "Thane" }
{ "_id" : ObjectId("63c026f71d07171adb80e9fa"), "name" : "Karan", "Roll_no" : 6, "Class" : "MSc_CS", "Address" : "Pune", "address" : "Mumbai" }
>
```

Command to delete a single document from collection

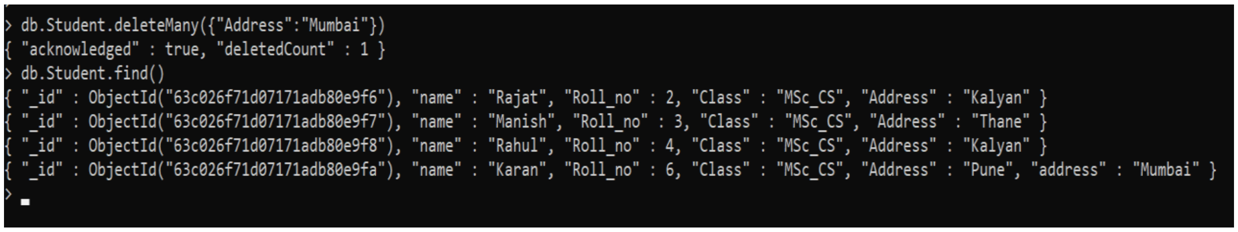
---->db.Student.deleteOne({"name":"Mohit"})



```
>
> db.Student.deleteOne({"name":"Mohit"})
{ "acknowledged" : true, "deletedCount" : 1 }
> db.Student.find()
{ "_id" : ObjectId("63c026d71d07171adb80e9f5"), "name" : "Raj", "Roll_no" : 1, "Class" : "MSc_CS", "Address" : "Mumbai" }
{ "_id" : ObjectId("63c026f71d07171adb80e9f6"), "name" : "Rajat", "Roll_no" : 2, "Class" : "MSc_CS", "Address" : "Kalyan" }
{ "_id" : ObjectId("63c026f71d07171adb80e9f7"), "name" : "Manish", "Roll_no" : 3, "Class" : "MSc_CS", "Address" : "Thane" }
{ "_id" : ObjectId("63c026f71d07171adb80e9f8"), "name" : "Rahul", "Roll_no" : 4, "Class" : "MSc_CS", "Address" : "Kalyan" }
{ "_id" : ObjectId("63c026f71d07171adb80e9fa"), "name" : "Karan", "Roll_no" : 6, "Class" : "MSc_CS", "Address" : "Pune", "address" : "Mumbai" }
>
```

Command to delete multiple documents from collection

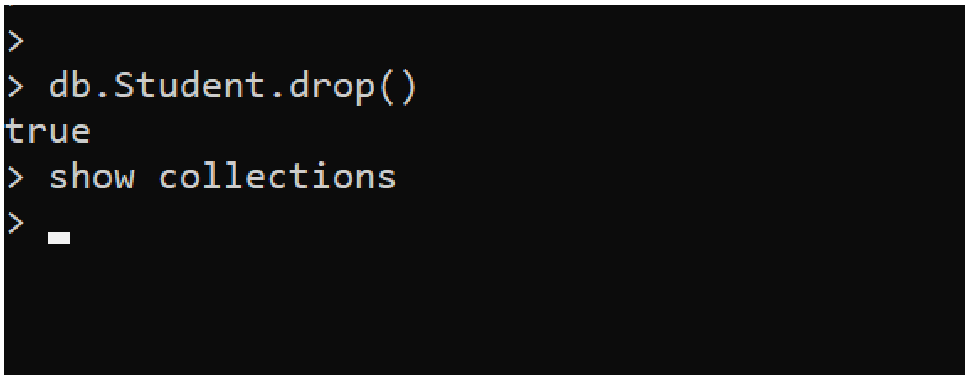
---->db.Student.deleteMany({"Address":"Mumbai"})



```
> db.Student.deleteMany({"Address":"Mumbai"})
{ "acknowledged" : true, "deletedCount" : 1 }
> db.Student.find()
{ "_id" : ObjectId("63c026f71d07171adb80e9f6"), "name" : "Rajat", "Roll_no" : 2, "Class" : "MSc_CS", "Address" : "Kalyan" }
{ "_id" : ObjectId("63c026f71d07171adb80e9f7"), "name" : "Manish", "Roll_no" : 3, "Class" : "MSc_CS", "Address" : "Thane" }
{ "_id" : ObjectId("63c026f71d07171adb80e9f8"), "name" : "Rahul", "Roll_no" : 4, "Class" : "MSc_CS", "Address" : "Kalyan" }
{ "_id" : ObjectId("63c026f71d07171adb80e9fa"), "name" : "Karan", "Roll_no" : 6, "Class" : "MSc_CS", "Address" : "Pune", "address" : "Mumbai" }
>
```

Command to drop a collection

---->db.Student.drop()



```
>
> db.Student.drop()
true
> show collections
>
```

Practical 7

Aim: Demonstrate the Accessing and Storing and performing CRUD operations in

1. HBase
2. Apache Cassandra

Background Information:

HBase:

HBase is an open-source non-relational distributed database modeled after Google's Bigtable and written in Java. It is developed as part of Apache Software Foundation's Apache Hadoop project and runs on top of HDFS (Hadoop Distributed File System) or Alluxio, providing Bigtable-like capabilities for Hadoop. That is, it provides a fault-tolerant way of storing large quantities of sparse data (small amounts of information caught within a large collection of empty or unimportant data, such as finding the 50 largest items in a group of 2 billion records or finding the non-zero items representing less than 0.1% of a huge collection).

Apache Cassandra:

Cassandra is a free and open-source, distributed, wide-column store, NoSQL database management system designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure. Cassandra offers support for clusters spanning multiple datacenters, with asynchronous masterless replication allowing low latency operations for all clients. Cassandra was designed to implement a combination of Amazon's Dynamo distributed storage and replication techniques combined with Google's Bigtable data and storage engine model.

Create Operation:

The create or insert operations are used to insert or add new documents in the collection. If a collection does not exist, then it will create a new collection in the database.

Read Operation:

The Read operations are used to retrieve documents from the collection, or in other words, read operations are used to query a collection for a document.

Update Operation:

The update operations are used to update or modify the existing document in the collection.

Delete Operation:

The delete operation is used to delete or remove documents from a collection.

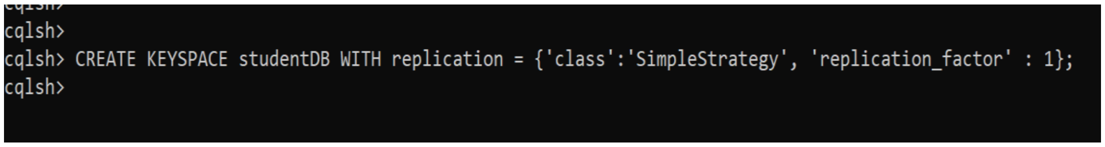
Code/Output:

1. HBase

2. Apache Cassandra

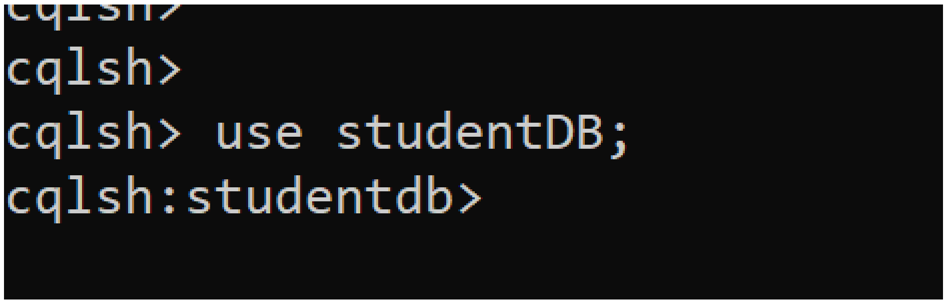
Command to create keyspace

```
---->CREATE KEYSPACE studentDB_WITH replication = {'class':'SimpleStrategy', 'replication_factor' : 1};
```


```
cqlsh>
cqlsh> CREATE KEYSPACE studentDB WITH replication = {'class':'SimpleStrategy', 'replication_factor' : 1};
cqlsh>
```

Command to use keyspace

---->use studentDB;


```
cqlsh>
cqlsh> use studentDB;
cqlsh:studentdb>
```

Command to create table

---->CREATE TABLE student(

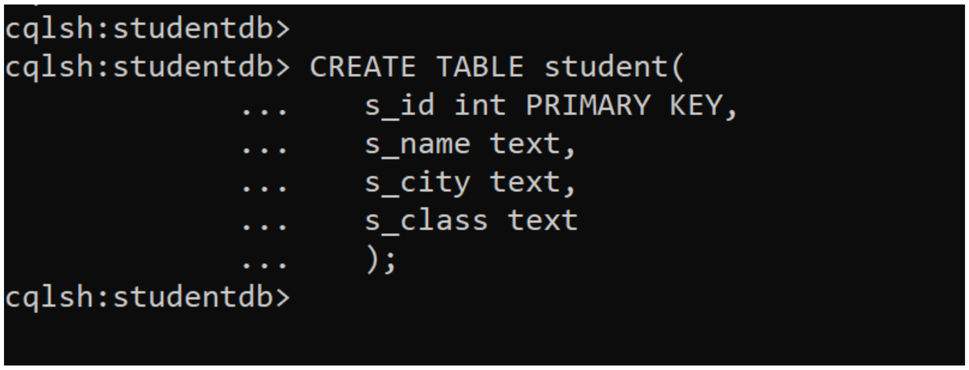
__s_id int PRIMARY KEY,

__s_name text,

__s_city text,

__s_class text

__);


```
cqlsh:studentdb>
cqlsh:studentdb> CREATE TABLE student(
             ...    s_id int PRIMARY KEY,
             ...    s_name text,
             ...    s_city text,
             ...    s_class text
             ...    );
cqlsh:studentdb>
```

Command to insert data

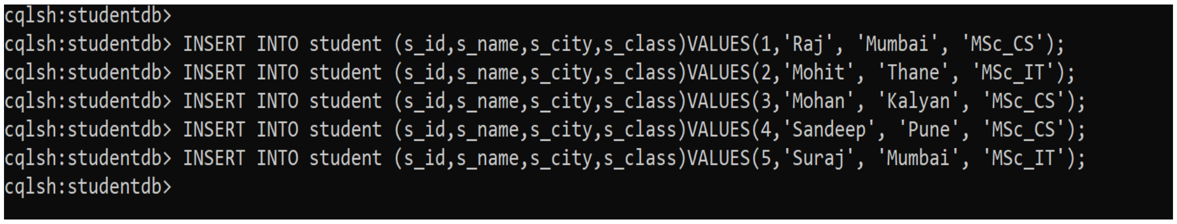
---->INSERT INTO student (s_id,s_name,s_city,s_class)VALUES(1,'Raj', 'Mumbai', 'MSc_CS');

---->INSERT INTO student (s_id,s_name,s_city,s_class)VALUES(2,'Mohit', 'Thane', 'MSc_IT');

---->INSERT INTO student (s_id,s_name,s_city,s_class)VALUES(3,'Mohan', 'Kalyan', 'MSc_CS');

---->INSERT INTO student (s_id,s_name,s_city,s_class)VALUES(4,'Sandeep', 'Pune', 'MSc_CS');

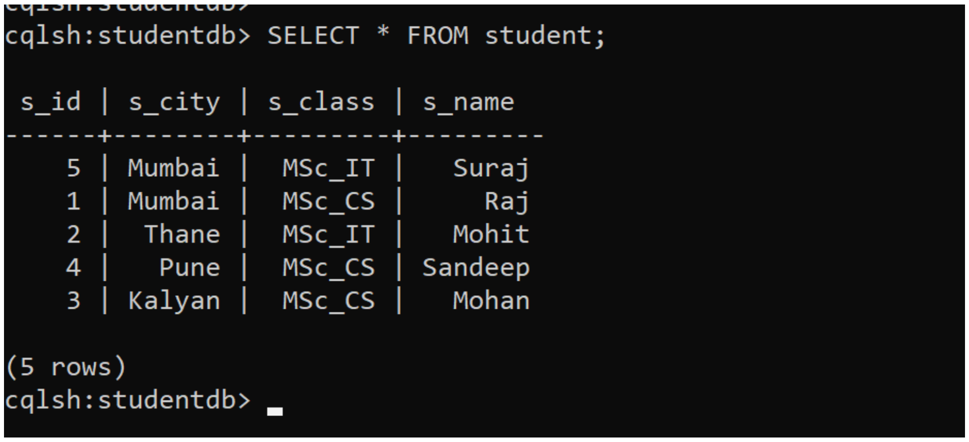
---->INSERT INTO student (s_id,s_name,s_city,s_class)VALUES(5,'Suraj', 'Mumbai', 'MSc_IT');



```
cqlsh:studentdb>
cqlsh:studentdb> INSERT INTO student (s_id,s_name,s_city,s_class)VALUES(1,'Raj', 'Mumbai', 'MSc_CS');
cqlsh:studentdb> INSERT INTO student (s_id,s_name,s_city,s_class)VALUES(2,'Mohit', 'Thane', 'MSc_IT');
cqlsh:studentdb> INSERT INTO student (s_id,s_name,s_city,s_class)VALUES(3,'Mohan', 'Kalyan', 'MSc_CS');
cqlsh:studentdb> INSERT INTO student (s_id,s_name,s_city,s_class)VALUES(4,'Sandeep', 'Pune', 'MSc_CS');
cqlsh:studentdb> INSERT INTO student (s_id,s_name,s_city,s_class)VALUES(5,'Suraj', 'Mumbai', 'MSc_IT');
cqlsh:studentdb>
```

Command to read data

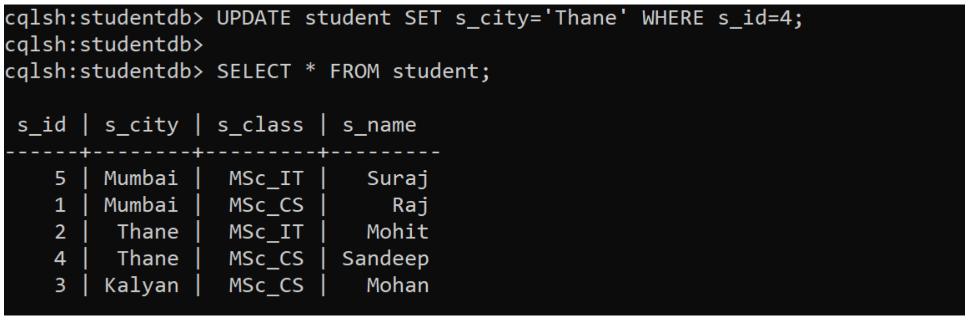
---->SELECT * FROM student;



```
cqlsh:studentdb> SELECT * FROM student;

 s_id | s_city | s_class | s_name
------+--------+---------+---------
    5 | Mumbai |  MSc_IT |   Suraj
    1 | Mumbai |  MSc_CS |     Raj
    2 |  Thane |  MSc_IT |   Mohit
    4 |   Pune |  MSc_CS | Sandeep
    3 | Kalyan |  MSc_CS |   Mohan

(5 rows)
cqlsh:studentdb>
```

Command to update data

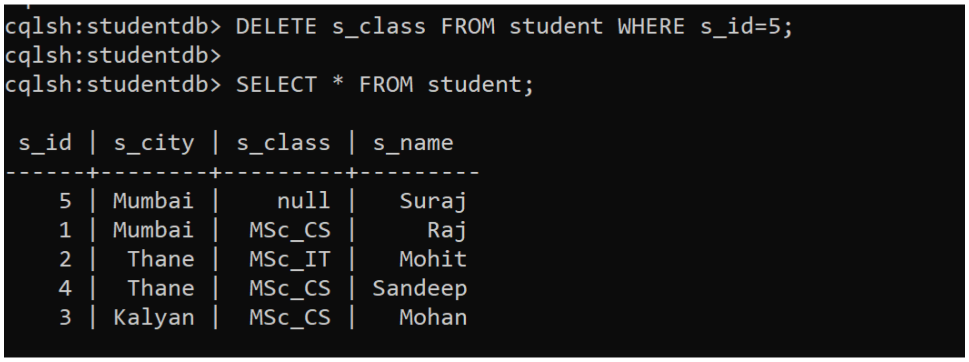
---->UPDATE student SET s_city='Thane' WHERE s_id=4;



```
cqlsh:studentdb> UPDATE student SET s_city='Thane' WHERE s_id=4;
cqlsh:studentdb>
cqlsh:studentdb> SELECT * FROM student;

 s_id | s_city | s_class | s_name
------+--------+---------+---------
    5 | Mumbai |  MSc_IT |   Suraj
    1 | Mumbai |  MSc_CS |     Raj
    2 |  Thane |  MSc_IT |   Mohit
    4 |  Thane |  MSc_CS | Sandeep
    3 | Kalyan |  MSc_CS |   Mohan
```

Command to delete a data

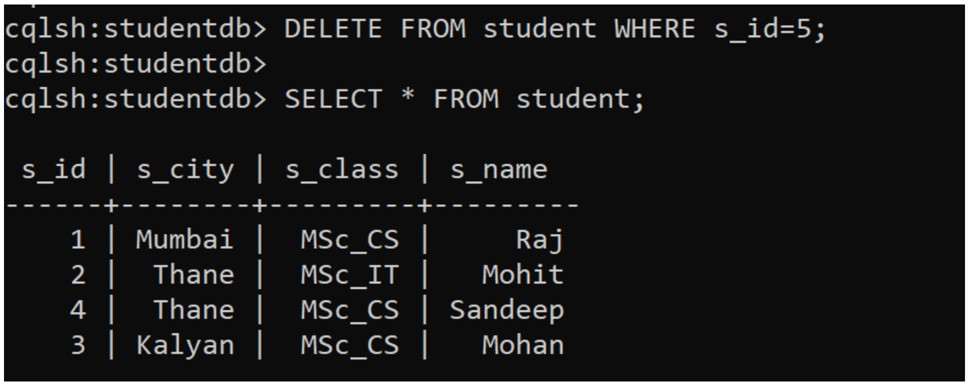
---->DELETE s_class FROM student WHERE s_id=5;



```
cqlsh:studentdb> DELETE s_class FROM student WHERE s_id=5;
cqlsh:studentdb>
cqlsh:studentdb> SELECT * FROM student;

 s_id | s_city | s_class | s_name
------+--------+---------+---------
    5 | Mumbai |    null |   Suraj
    1 | Mumbai |  MSc_CS |     Raj
    2 |  Thane |  MSc_IT |   Mohit
    4 |  Thane |  MSc_CS | Sandeep
    3 | Kalyan |  MSc_CS |   Mohan
```

Command to delete a row

---->DELETE FROM student WHERE s_id=5;



```
cqlsh:studentdb> DELETE FROM student WHERE s_id=5;
cqlsh:studentdb>
cqlsh:studentdb> SELECT * FROM student;

 s_id | s_city | s_class | s_name
------+--------+---------+---------
    1 | Mumbai |  MSc_CS |     Raj
    2 |  Thane |  MSc_IT |   Mohit
    4 |  Thane |  MSc_CS | Sandeep
    3 | Kalyan |  MSc_CS |   Mohan
```

Practical 8

Aim: Demonstrating MapReduce in MongoDB to count the number of female (F) and male (M) respondents in the database.

Background Information:

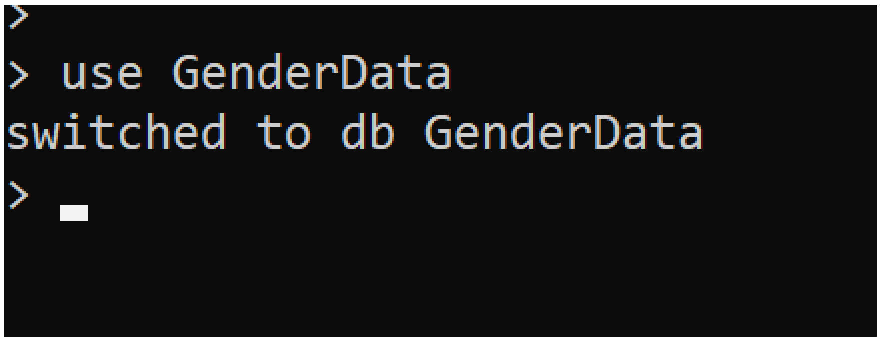
MapReduce:

In MongoDB, map-reduce is a data processing programming model that helps to perform operations on large data sets and produce aggregated results. MongoDB provides the mapReduce() function to perform the map-reduce operations. This function has two main functions, i.e., map function and reduce function. The map function is used to group all the data based on the key-value and the reduce function is used to perform operations on the mapped data. So, the data is independently mapped and reduced in different spaces and then combined in the function and the result will be saved to the specified new collection. This mapReduce() function generally operates on large data sets only. Using Map Reduce you can perform aggregation operations such as max, avg on the data using some key and it is like groupBy in SQL. It performs on data independently and parallel.

Code/Output:

Command to create a Database

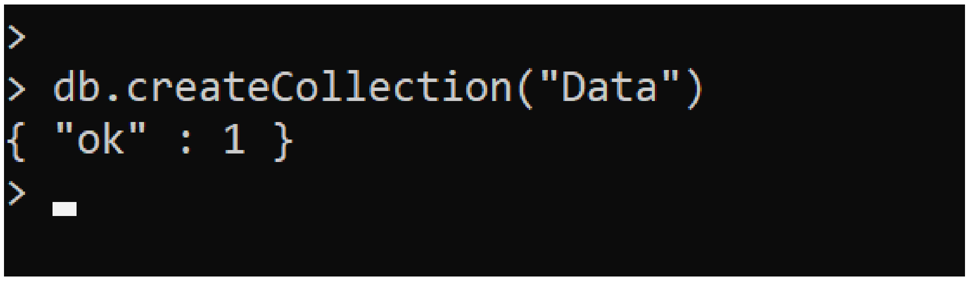
---->use GenderData



```
>
> use GenderData
switched to db GenderData
>
```

Command to create a Collection

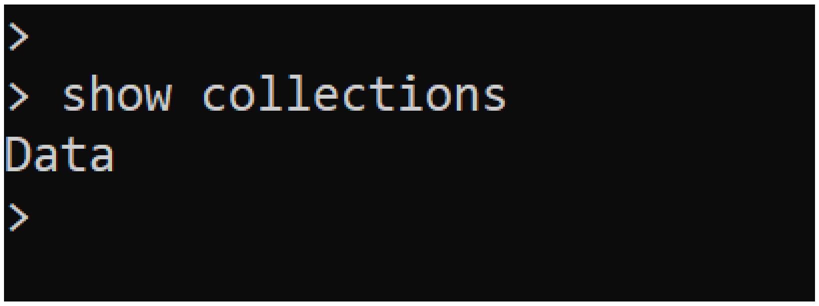
---->db.createCollection("Data")



```
>
> db.createCollection("Data")
{ "ok" : 1 }
>
```

Command to view Collection

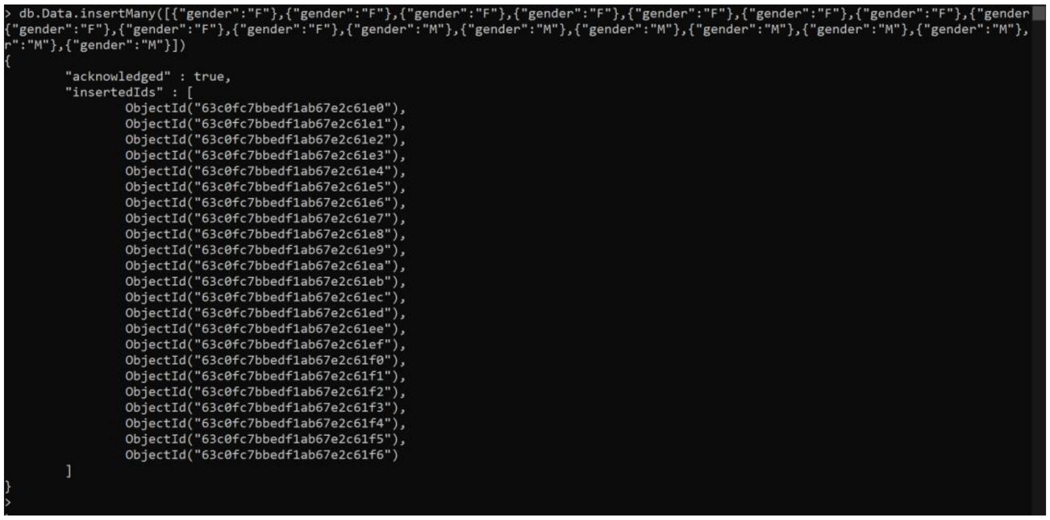
---->show collections



```
>
> show collections
Data
>
```

Command to insert multiple documents in Collection

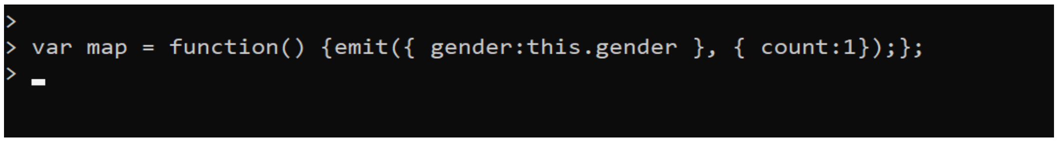
>db.Data.insertMany([{"gender":"F"},{"gender":"F"},{"gender":"F"},{"gender":"F"},{"gender":"F"},{"gender":"F"},{"gender":"F"},{"gender":"F"},{"gender":"F"},{"gender":"F"},{"gender":"F"},{"gender":"F"},{"gender":"F"},{"gender":"M"},{"gender":"M"},{"gender":"M"},{"gender":"M"},{"gender":"M"},{"gender":"M"},{"gender":"M"},{"gender":"M"},{"gender":"M"},{"gender":"M"}])



```
> db.Data.insertMany([{"gender":"F"},{"gender":"F"},{"gender":"F"},{"gender":"F"},{"gender":"F"},{"gender":"F"},{"gender":"F"},{"gender
{"gender":"F"},{"gender":"F"},{"gender":"F"},{"gender":"M"},{"gender":"M"},{"gender":"M"},{"gender":"M"},{"gender":"M"},{"gender":"M"},
r":"M"},{"gender":"M"}])
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("63c0fc7bbedf1ab67e2c61e0"),
                ObjectId("63c0fc7bbedf1ab67e2c61e1"),
                ObjectId("63c0fc7bbedf1ab67e2c61e2"),
                ObjectId("63c0fc7bbedf1ab67e2c61e3"),
                ObjectId("63c0fc7bbedf1ab67e2c61e4"),
                ObjectId("63c0fc7bbedf1ab67e2c61e5"),
                ObjectId("63c0fc7bbedf1ab67e2c61e6"),
                ObjectId("63c0fc7bbedf1ab67e2c61e7"),
                ObjectId("63c0fc7bbedf1ab67e2c61e8"),
                ObjectId("63c0fc7bbedf1ab67e2c61e9"),
                ObjectId("63c0fc7bbedf1ab67e2c61ea"),
                ObjectId("63c0fc7bbedf1ab67e2c61eb"),
                ObjectId("63c0fc7bbedf1ab67e2c61ec"),
                ObjectId("63c0fc7bbedf1ab67e2c61ed"),
                ObjectId("63c0fc7bbedf1ab67e2c61ee"),
                ObjectId("63c0fc7bbedf1ab67e2c61ef"),
                ObjectId("63c0fc7bbedf1ab67e2c61f0"),
                ObjectId("63c0fc7bbedf1ab67e2c61f1"),
                ObjectId("63c0fc7bbedf1ab67e2c61f2"),
                ObjectId("63c0fc7bbedf1ab67e2c61f3"),
                ObjectId("63c0fc7bbedf1ab67e2c61f4"),
                ObjectId("63c0fc7bbedf1ab67e2c61f5"),
                ObjectId("63c0fc7bbedf1ab67e2c61f6")
        ]
}
>
```

Command to create map

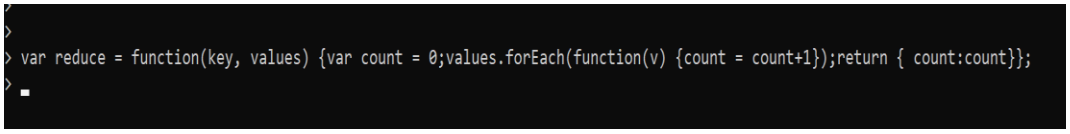
---->var map = function() {emit({ gender:this.gender }, { count:1});};



```
>
> var map = function() {emit({ gender:this.gender }, { count:1});};
>
```

Command to create reduce

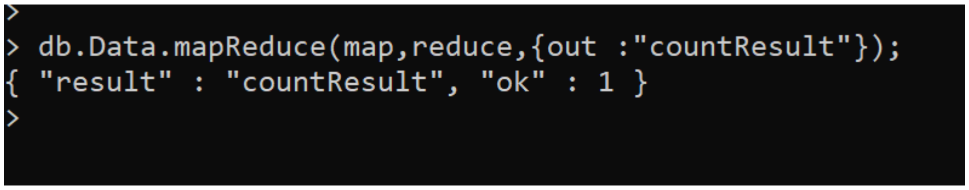
---->var reduce = function(key, values) {var count = 0;values.forEach(function(v) {count = count+1});return { count:count}};



```
>
> var reduce = function(key, values) {var count = 0;values.forEach(function(v) {count = count+1});return { count:count}};
>
```

Command to store output in new collection

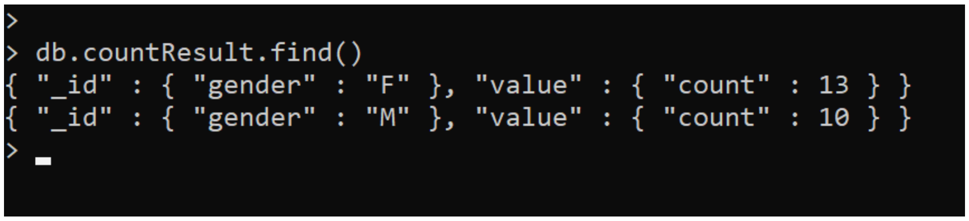
---->db.Data.mapReduce(map,reduce,{out :"countResult"});



```
>
> db.Data.mapReduce(map,reduce,{out :"countResult"});
{ "result" : "countResult", "ok" : 1 }
>
```

Command to view final count result

---->db.countResult.find()



```
>
> db.countResult.find()
{ "_id" : { "gender" : "F" }, "value" : { "count" : 13 } }
{ "_id" : { "gender" : "M" }, "value" : { "count" : 10 } }
>
```

Practical 9

Aim: Demonstrate the indexing and ordering operations in

1. MongoDB
2. CouchDB
3. Apache Cassandra

Background Information:

MongoDB:

MongoDB is a source-available cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with optional schemas. MongoDB is developed by MongoDB Inc. and licensed under the Server-Side Public License (SSPL) which is deemed non-free by several distributions.

CouchDB:

Apache CouchDB is an open-source document-oriented NoSQL database, implemented in Erlang. CouchDB uses multiple formats and protocols to store, transfer, and process its data. It uses JSON to store data, JavaScript as its query language using MapReduce, and HTTP for an API. CouchDB was first released in 2005 and later became an Apache Software Foundation project in 2008. Unlike a relational database, a CouchDB database does not store data and relationships in tables. Instead, each database is a collection of independent documents. Each document maintains its own data and self-contained schema. An application may access multiple databases, such as one stored on a user's mobile phone and another on a server. Document metadata contains revision information, making it possible to merge any differences that may have occurred while the databases were disconnected.

Apache Cassandra:

Cassandra is a free and open-source, distributed, wide-column store, NoSQL database management system designed to handle large amounts of data across many commodity servers, providing high availability with no single point of failure. Cassandra offers support for clusters spanning multiple datacenters, with asynchronous masterless replication allowing low latency operations for all clients. Cassandra was designed to implement a combination of Amazon's Dynamo distributed storage and replication techniques combined with Google's Bigtable data and storage engine model.

Indexing:

Indexing is used to optimize the performance of a database by minimizing the number of disk accesses required when a query is processed. The index is a type of data structure. It is used to locate and access the data in a database table quickly.

Sorting in SQL:

Sorting helps to sort the records that are retrieved. By default, it displays the records in ascending order of primary key. If we need to sort it based on different columns, then we need to specify it in ORDER BY clause. If we need to order by descending order, then DESC keyword must be added after the column list.

Sorting in MongoDB:

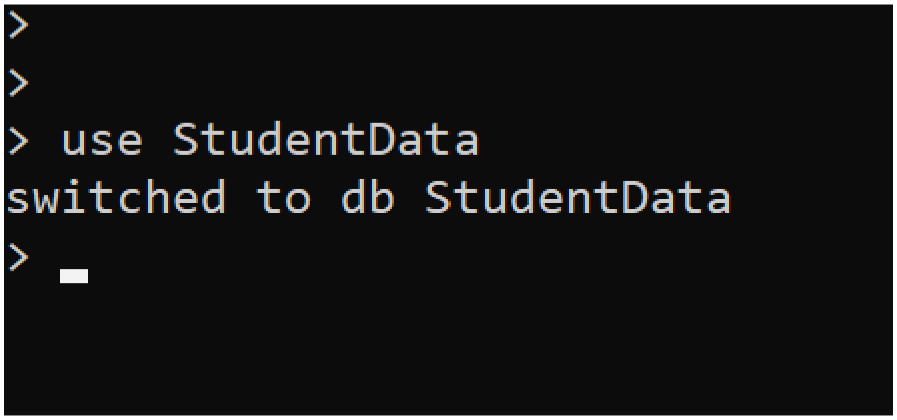
To sort documents in MongoDB, you need to use the sort () method. The method accepts a document containing a list of fields along with their sorting order. To specify sorting order 1 and -1 are used. 1 is used for ascending order while -1 is used for descending order.

<u>Code/Output:</u>

<u>1. MongoDB</u>

Command to create a Database

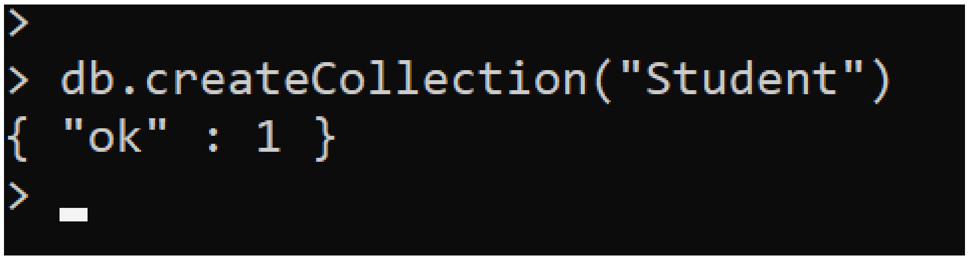
---->use StudentData



```
>
>
> use StudentData
switched to db StudentData
> 
```

Command to create a Collection

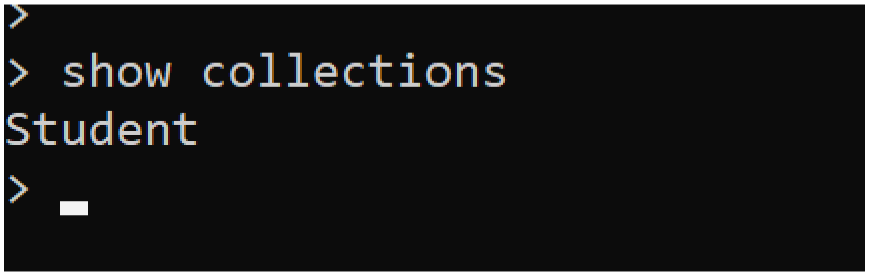
---->db.createCollection("Student")



```
>
> db.createCollection("Student")
{ "ok" : 1 }
> 
```

Command to view Collection

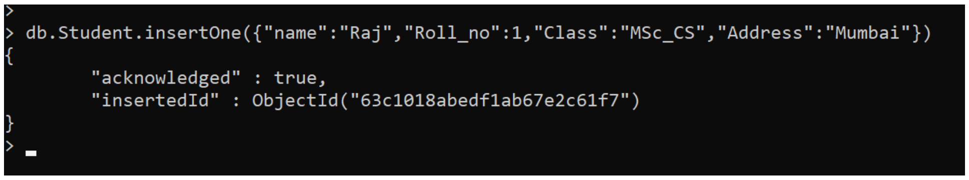
---->show collections



```
>
> show collections
Student
> 
```

Command to insert a single document in Collection

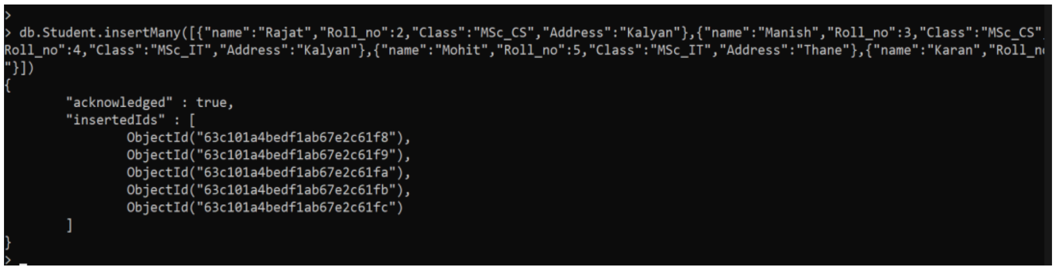
---->db.Student.insertOne({"name":"Raj","Roll_no":1,"Class":"MSc_CS","Address":"Mumbai"})



```
>
> db.Student.insertOne({"name":"Raj","Roll_no":1,"Class":"MSc_CS","Address":"Mumbai"})
{
        "acknowledged" : true,
        "insertedId" : ObjectId("63c1018abedf1ab67e2c61f7")
}
> 
```

Command to insert multiple documents in Collection

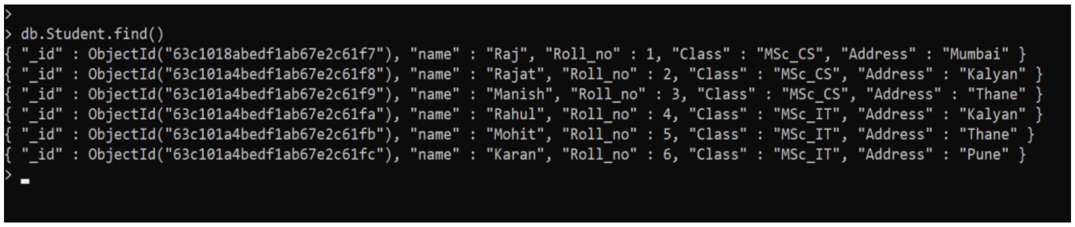
>db.Student.insertMany([{"name":"Rajat","Roll_no":2,"Class":"MSc_CS","Address":"Kalyan"},{"name":"Manish","Roll_no":3,"Class":"MSc_CS","Address":"Thane"},{"name":"Rahul","Roll_no":4,"Class":"MSc_IT","Address":"Kalyan"},{"name":"Mohit","Roll_no":5,"Class":"MSc_IT","Address":"Thane"},{"name":"Karan","Roll_no":6,"Class":"MSc_IT","Address":"Pune"}])



```
>
> db.Student.insertMany([{"name":"Rajat","Roll_no":2,"Class":"MSc_CS","Address":"Kalyan"},{"name":"Manish","Roll_no":3,"Class":"MSc_CS"
Roll_no":4,"Class":"MSc_IT","Address":"Kalyan"},{"name":"Mohit","Roll_no":5,"Class":"MSc_IT","Address":"Thane"},{"name":"Karan","Roll_n
"}])
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("63c101a4bedf1ab67e2c61f8"),
                ObjectId("63c101a4bedf1ab67e2c61f9"),
                ObjectId("63c101a4bedf1ab67e2c61fa"),
                ObjectId("63c101a4bedf1ab67e2c61fb"),
                ObjectId("63c101a4bedf1ab67e2c61fc")
        ]
}
> 
```

Command to view all documents

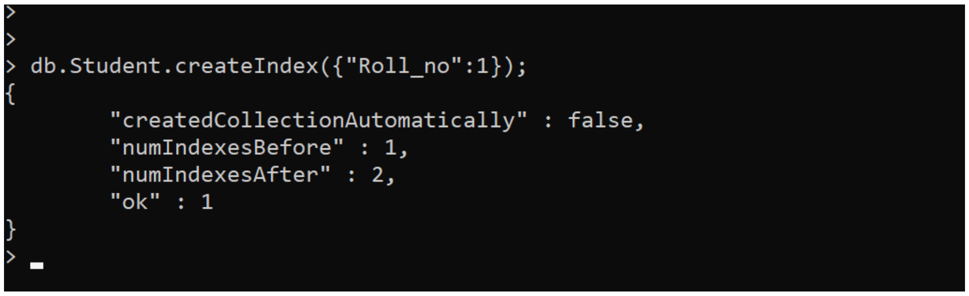
---->db.Student.find()



```
>
> db.Student.find()
{ "_id" : ObjectId("63c1018abedf1ab67e2c61f7"), "name" : "Raj", "Roll_no" : 1, "Class" : "MSc_CS", "Address" : "Mumbai" }
{ "_id" : ObjectId("63c101a4bedf1ab67e2c61f8"), "name" : "Rajat", "Roll_no" : 2, "Class" : "MSc_CS", "Address" : "Kalyan" }
{ "_id" : ObjectId("63c101a4bedf1ab67e2c61f9"), "name" : "Manish", "Roll_no" : 3, "Class" : "MSc_CS", "Address" : "Thane" }
{ "_id" : ObjectId("63c101a4bedf1ab67e2c61fa"), "name" : "Rahul", "Roll_no" : 4, "Class" : "MSc_IT", "Address" : "Kalyan" }
{ "_id" : ObjectId("63c101a4bedf1ab67e2c61fb"), "name" : "Mohit", "Roll_no" : 5, "Class" : "MSc_IT", "Address" : "Thane" }
{ "_id" : ObjectId("63c101a4bedf1ab67e2c61fc"), "name" : "Karan", "Roll_no" : 6, "Class" : "MSc_IT", "Address" : "Pune" }
> 
```

Command to Create Index

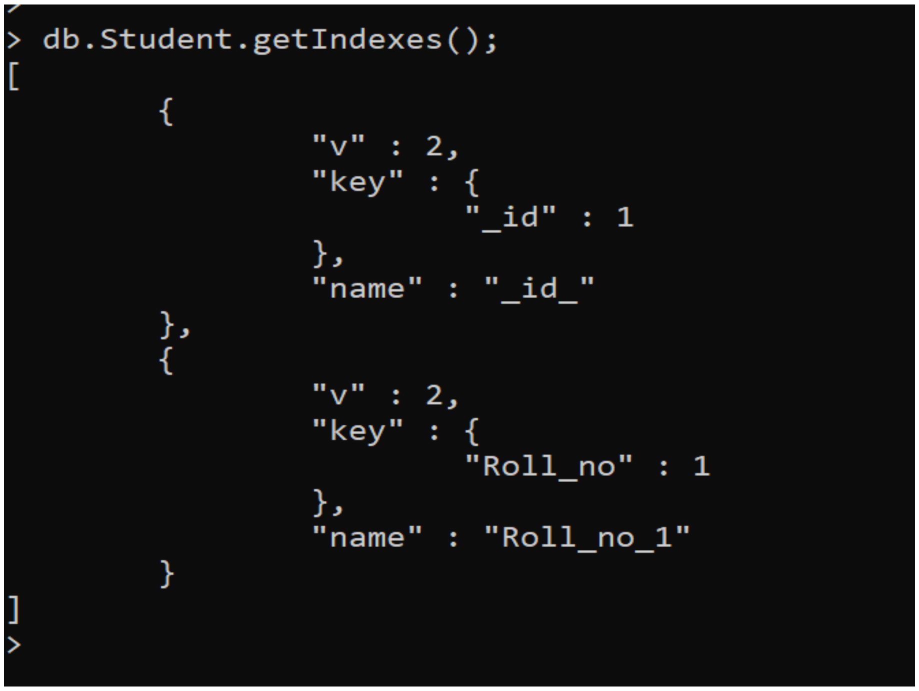
---->db.Student.createIndex({"Roll_no":1});



```
>
>
> db.Student.createIndex({"Roll_no":1});
{
        "createdCollectionAutomatically" : false,
        "numIndexesBefore" : 1,
        "numIndexesAfter" : 2,
        "ok" : 1
}
> 
```

Command to view all indexes in collection

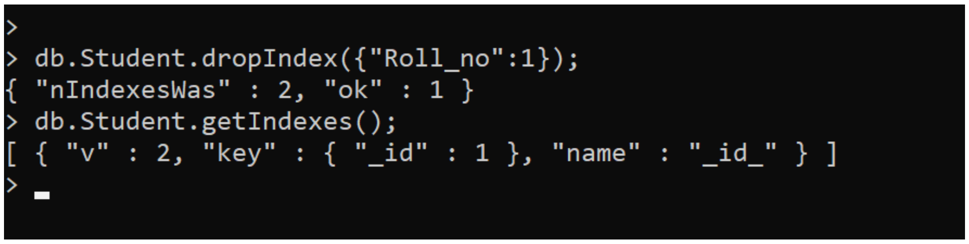
---->db.Student.getIndexes();



```
> db.Student.getIndexes();
[
        {
                "v" : 2,
                "key" : {
                        "_id" : 1
                },
                "name" : "_id_"
        },
        {
                "v" : 2,
                "key" : {
                        "Roll_no" : 1
                },
                "name" : "Roll_no_1"
        }
]
>
```

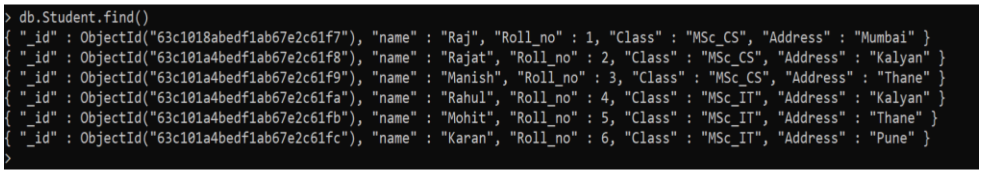
Command to Drop Index

---->db.Student.dropIndex({"Roll_no":1});



```
>
> db.Student.dropIndex({"Roll_no":1});
{ "nIndexesWas" : 2, "ok" : 1 }
> db.Student.getIndexes();
[ { "v" : 2, "key" : { "_id" : 1 }, "name" : "_id_" } ]
> 
```

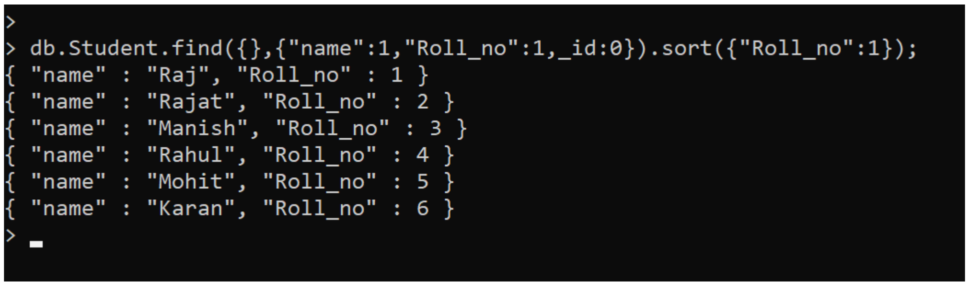
Before Sorting/Ordering



```
> db.Student.find()
{ "_id" : ObjectId("63c1018abedf1ab67e2c61f7"), "name" : "Raj", "Roll_no" : 1, "Class" : "MSc_CS", "Address" : "Mumbai" }
{ "_id" : ObjectId("63c101a4bedf1ab67e2c61f8"), "name" : "Rajat", "Roll_no" : 2, "Class" : "MSc_CS", "Address" : "Kalyan" }
{ "_id" : ObjectId("63c101a4bedf1ab67e2c61f9"), "name" : "Manish", "Roll_no" : 3, "Class" : "MSc_CS", "Address" : "Thane" }
{ "_id" : ObjectId("63c101a4bedf1ab67e2c61fa"), "name" : "Rahul", "Roll_no" : 4, "Class" : "MSc_IT", "Address" : "Kalyan" }
{ "_id" : ObjectId("63c101a4bedf1ab67e2c61fb"), "name" : "Mohit", "Roll_no" : 5, "Class" : "MSc_IT", "Address" : "Thane" }
{ "_id" : ObjectId("63c101a4bedf1ab67e2c61fc"), "name" : "Karan", "Roll_no" : 6, "Class" : "MSc_IT", "Address" : "Pune" }
>
```

Command to order data in ascending order

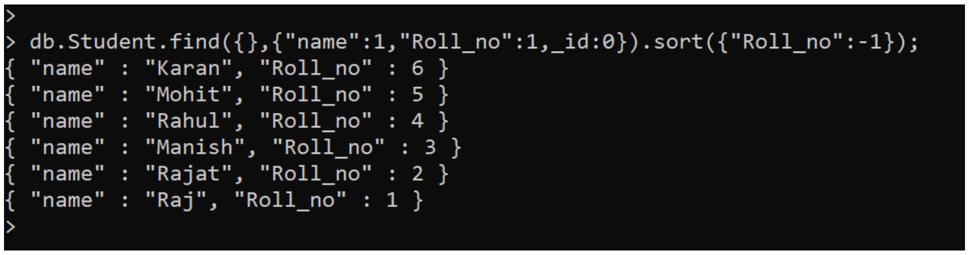
---->db.Student.find({},{"name":1,"Roll_no":1,_id:0}).sort({"Roll_no":1});



```
>
> db.Student.find({},{"name":1,"Roll_no":1,_id:0}).sort({"Roll_no":1});
{ "name" : "Raj", "Roll_no" : 1 }
{ "name" : "Rajat", "Roll_no" : 2 }
{ "name" : "Manish", "Roll_no" : 3 }
{ "name" : "Rahul", "Roll_no" : 4 }
{ "name" : "Mohit", "Roll_no" : 5 }
{ "name" : "Karan", "Roll_no" : 6 }
>
```

Command to order data in descending order

---->db.Student.find({},{"name":1,"Roll_no":1,_id:0}).sort({"Roll_no":-1});



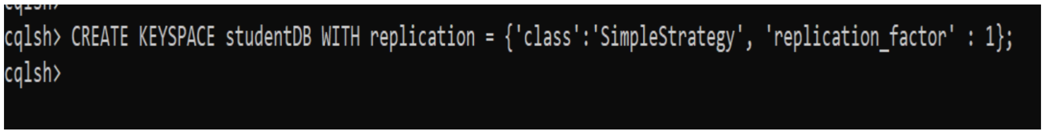
```
>
> db.Student.find({},{"name":1,"Roll_no":1,_id:0}).sort({"Roll_no":-1});
{ "name" : "Karan", "Roll_no" : 6 }
{ "name" : "Mohit", "Roll_no" : 5 }
{ "name" : "Rahul", "Roll_no" : 4 }
{ "name" : "Manish", "Roll_no" : 3 }
{ "name" : "Rajat", "Roll_no" : 2 }
{ "name" : "Raj", "Roll_no" : 1 }
>
```

2. CouchDB

3. Apache Cassandra

Command to create keyspace

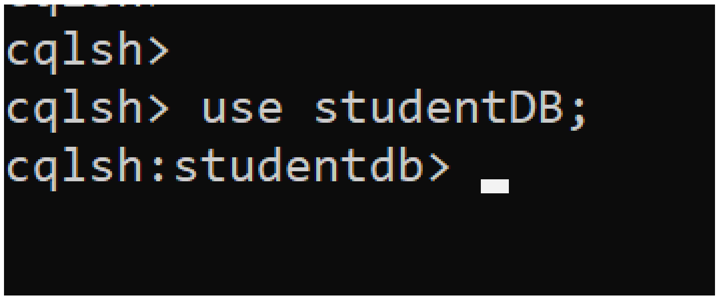
---->CREATE KEYSPACE studentDB WITH replication = {'class':'SimpleStrategy', 'replication_factor' : 1};



```
cqlsh> CREATE KEYSPACE studentDB WITH replication = {'class':'SimpleStrategy', 'replication_factor' : 1};
cqlsh>
```

Command to use keyspace

---->use studentDB;



```
cqlsh>
cqlsh> use studentDB;
cqlsh:studentdb>
```

Command to create table

---->CREATE TABLE student(

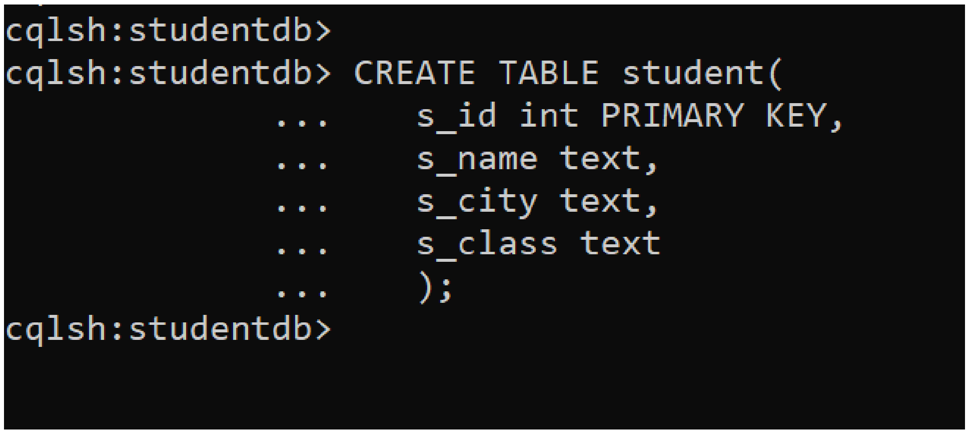
_s_id int PRIMARY KEY,

__s_name text,

__s_city text,

__s_class text

__);



```
cqlsh:studentdb>
cqlsh:studentdb> CREATE TABLE student(
             ...    s_id int PRIMARY KEY,
             ...    s_name text,
             ...    s_city text,
             ...    s_class text
             ...    );
cqlsh:studentdb>
```

Command to insert data

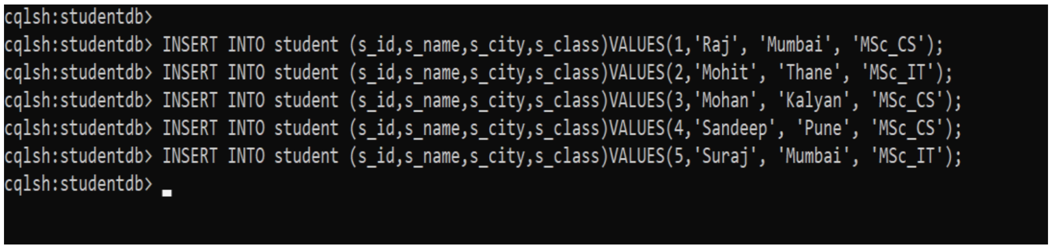
---->INSERT INTO student (s_id,s_name,s_city,s_class)VALUES(1,'Raj', 'Mumbai', 'MSc_CS');

---->INSERT INTO student (s_id,s_name,s_city,s_class)VALUES(2,'Mohit', 'Thane', 'MSc_IT');

---->INSERT INTO student (s_id,s_name,s_city,s_class)VALUES(3,'Mohan', 'Kalyan', 'MSc_CS');

---->INSERT INTO student (s_id,s_name,s_city,s_class)VALUES(4,'Sandeep', 'Pune', 'MSc_CS');

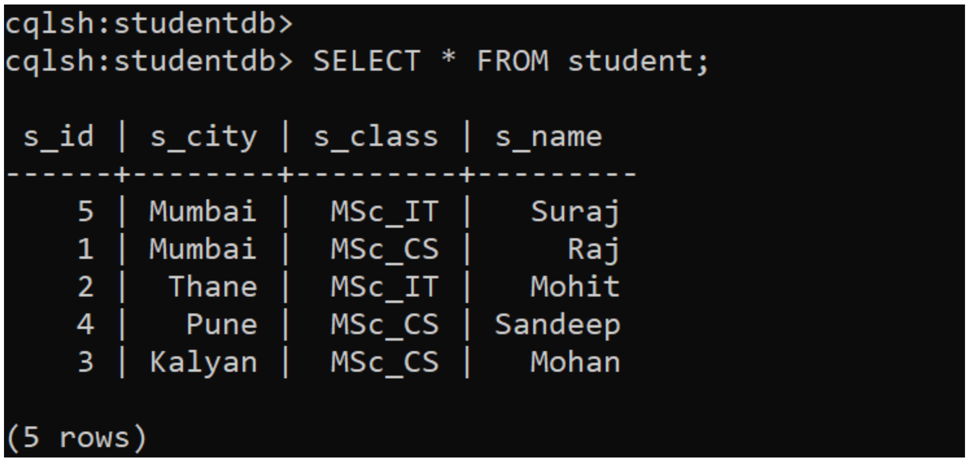
---->INSERT INTO student (s_id,s_name,s_city,s_class)VALUES(5,'Suraj', 'Mumbai', 'MSc_IT');



```
cqlsh:studentdb>
cqlsh:studentdb> INSERT INTO student (s_id,s_name,s_city,s_class)VALUES(1,'Raj', 'Mumbai', 'MSc_CS');
cqlsh:studentdb> INSERT INTO student (s_id,s_name,s_city,s_class)VALUES(2,'Mohit', 'Thane', 'MSc_IT');
cqlsh:studentdb> INSERT INTO student (s_id,s_name,s_city,s_class)VALUES(3,'Mohan', 'Kalyan', 'MSc_CS');
cqlsh:studentdb> INSERT INTO student (s_id,s_name,s_city,s_class)VALUES(4,'Sandeep', 'Pune', 'MSc_CS');
cqlsh:studentdb> INSERT INTO student (s_id,s_name,s_city,s_class)VALUES(5,'Suraj', 'Mumbai', 'MSc_IT');
cqlsh:studentdb>
```

Command to read data

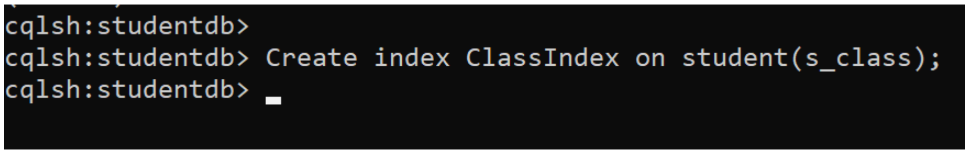
---->SELECT * FROM student;



```
cqlsh:studentdb>
cqlsh:studentdb> SELECT * FROM student;

 s_id | s_city | s_class | s_name
------+--------+---------+---------
    5 | Mumbai |  MSc_IT |   Suraj
    1 | Mumbai |  MSc_CS |     Raj
    2 |  Thane |  MSc_IT |   Mohit
    4 |   Pune |  MSc_CS | Sandeep
    3 | Kalyan |  MSc_CS |   Mohan

(5 rows)
```

Command to create index

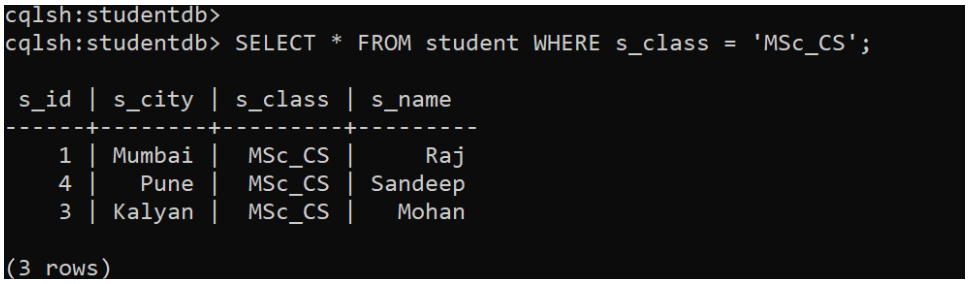
---->Create index ClassIndex on student(s_class);



```
cqlsh:studentdb>
cqlsh:studentdb> Create index ClassIndex on student(s_class);
cqlsh:studentdb> _
```

Command to view filtered data

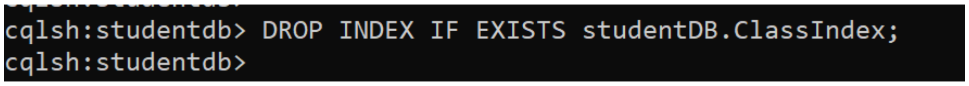
---->SELECT * FROM student WHERE s_class = 'MSc_CS';



```
cqlsh:studentdb>
cqlsh:studentdb> SELECT * FROM student WHERE s_class = 'MSc_CS';

 s_id | s_city | s_class | s_name
------+--------+---------+---------
    1 | Mumbai |  MSc_CS |     Raj
    4 |   Pune |  MSc_CS | Sandeep
    3 | Kalyan |  MSc_CS |   Mohan

(3 rows)
```

Command to delete index

---->DROP INDEX IF EXISTS studentDB.ClassIndex;



```
cqlsh:studentdb> DROP INDEX IF EXISTS studentDB.ClassIndex;
cqlsh:studentdb>
```