VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT on

BIG DATA ANALYTICS
(20CS6PEBDA)

Submitted by

Mohammed Ibrahim Rahil S (1BM19CS090)

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING BENGALURU-560019
May-2022 to July-2022
(Autonomous Institution under VTU)

B. M. S. College of Engineering,

Bull Temple Road, Bangalore 560019

(Affiliated To Visvesvaraya Technological University, Belgaum)

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the Lab work entitled "**BIG DATA ANALYTICS**" was carried out by **Mohammed Ibrahim Rahil S (1BM19CS090),** who is a bona fide student of **B. M. S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2022. The Lab report has been approved as it satisfies the academic requirements in respect of the course **BIG DATA ANALYTICS (20CS6PEBDA)** work prescribed for the said degree.

Dr. Pallavi G B
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyoti S Nayak
Head pf Department
Department of CSE
BMSCE, Bengaluru

`

Index Sheet

Course Outcome

CO1	Apply the concept of NoSQL, Hadoop or Spark for a given task
CO2	Analyze the Big Data and obtain insight using data analytics mechanisms.
CO3	Design and implement Big data applications by applying NoSQL, Hadoop or Spark

MongoDB Lab -1

WORKING WITH MONGODB

I. CREATE DATABASE IN MONGODB.

use myDB;

Confirm the existence of your database

```
> use naman
switched to db naman
```

db;

To list all databases

show dbs;

```
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
```

II. CRUD (CREATE, READ, UPDATE, DELETE) OPERATIONS

1. To create a collection by the name "Student". Let us take a look at the collection list prior to the creation of the new collection "Student".

   db.createCollection("Student"); **=>** *sql equivalent* **CREATE TABLE STUDENT(...);**

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

2. To drop a collection by the name “Student”.

```
db.Student.drop();
```

```
> db.Student.drop();
true
```

3. Create a collection by the name “Students” and store the following data in it.

```
db.Student.insert({_id:1,StudName:"MichelleJacintha",Grade:"VII",Hobbies:"InternetSurfing"});
```

4. Insert the document for “AryanDavid” in to the Students collection only if it does not already exist in the collection. However, if it is already present in the collection, then update the document with new values. (Update his Hobbies from “Skating” to “Chess”.) Use “Update else insert” (if there is an existing document, it will attempt to update it, if there is no existing document then it will insert it).

```
db.Student.update({_id:3,StudName:"AryanDavid",Grade:"VII"},{$set:{Hobbies:"Skating"}},{upsert:true});
```

```
>
db.Student.update({_id:3,StudName:"abc",Grade:"VII"},{$set:{Hobbies:"Skating"}},{upsert:true});
WriteResult({ "nMatched" : 0, "nUpserted" : 1, "nModified" : 0, "_id" : 3 })
```

```
> db.Student.update({_id:3},{$set:{Hobbies:"Chess"}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

5. FIND METHOD

A. To search for documents from the "Students" collection based on certain search criteria.

```
db.Student.find({StudName:"Aryan David"});
({cond..},{columns.. column:1, columnname:0}  )
```

```
> db.Student.find({StudName:"abc"});
{ "_id" : 3, "Grade" : "VII", "StudName" : "abc", "Hobbies" : "Skating" }
```

B. To display only the StudName and Grade from all the documents of the Students collection. The identifier_id should be suppressed and NOT displayed.

```
db.Student.find({},{StudName:1,Grade:1,_id:0});
```

```
> db.Student.find({},{StudName:1,Grade:1,_id:0});>
db.food.find({fruits:['grapes','mango','apple']}).pretty();
{ "_id" : 1, "fruits" : [ "grapes", "mango", "apple" ] }
{ "StudName" : "Naman", "Grade" : "VII" }
{ "Grade" : "VII", "StudName" : "abc" }
```

```
> db.food.find({fruits:['grapes','mango','apple']}).pretty();>
db.food.find({fruits:['grapes','mango','apple']}).pretty();
{ "_id" : 1, "fruits" : [ "grapes", "mango", "apple" ] }
{ "_id" : 1, "fruits" : [ "grapes", "mango", "apple" ] }
```

C. To find those documents where the Grade is set to 'VII'

db.Student.find({Grade:{$eq:'VII'}}).pretty();

```
> db.Student.find({Grade:{$eq:&apos;VII&apos;}}).pretty();
{
        "_id" : 1,
        "StudName" : "Naman",
        "Grade" : "VII",
        "Hobbies" : "InternetSurfing"
}
{ "_id" : 3, "Grade" : "VII", "StudName" : "abc", "Hobbies" : "Chess" }
```

D. To find those documents from the Students collection where the Hobbies is set to either 'Chess' or is set to 'Skating'.

db.Student.find({Hobbies :{ $in: ['Chess','Skating']}}).pretty ();

> db.Student.find({Hobbies:{$in:['Chess','Skating']}});

```
{ "_id" : 3, "Grade" : "VII", "StudName" : "abc", "Hobbies" : "Chess" }
```

E. To find documents from the Students collection where the StudName begins with "M".

db.Student.find({StudName:/^M/}).pretty();

```
> db.Student.find({StudName:/^N/}).pretty();
{
        "_id" : 1,
        "StudName" : "Naman",
        "Grade" : "VII",
        "Hobbies" : "InternetSurfing"
}
```

F. To find documents from the Students collection where the StudNamehas an "e" in any position.

db.Student.find({StudName:/e/}).pretty();

> db.Student.find({StudName:/m/}).pretty();

```
{
        "_id" : 1,
        "StudName" : "Naman",
        "Grade" : "VII",
        "Hobbies" : "InternetSurfing"
}
```

G. To find the number of documents in the Students collection.

db.Student.count();

```
> db.Student.count();
```

H. To sort the documents from the Students collection in the descending order of StudName.

db.Student.find().sort({StudName:-1}).pretty();

{ "_id" : 3, "Grade" : "VII", "StudName" : "abc", "Hobbies" : "Chess" }


```
{
        "_id" : 1,
        "StudName" : "Naman",
        "Grade" : "VII",
        "Hobbies" : "InternetSurfing"
}
```


II. Import data from a CSV file

Given a CSV file "sample.txt" in the D:drive, import the file into the MongoDB collection, "SampleJSON". The collection is in the database "test".

mongoimport --db Student --collection airlines --type csv –headerline --file /home/hduser/Desktop/airline.csv

V. Export data to a CSV file

This command used at the command prompt exports MongoDB JSON documents from "Customers" collection in the "test" database into a CSV file "Output.txt" in the D:drive.

mongoexport --host localhost --db Student --collection airlines --csv --out /home/hduser/Desktop/output.txt –fields "Year","Quarter"

V. Save Method :

V. Save Method :

Save() method will insert a new document, if the document with the _id does not exist. If it exists it will replace the exisiting document.

```
db.Students.save({StudName:"Vamsi", Grade:"VI"})
```

```
> db.Student.save({_id:3,StudName:"abc",Grade:"VI"});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

VI. Add a new field to existing Document:

```
db.Students.update({_id:4},{$set:{Location:"Network"}})
```

```
> db.Student.update({_id:1},{$set:{Location:"Network"}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

VII. Remove the field in an existing Document

```
db.Students.update({_id:4},{$unset:{Location:"Network"}})
```

```
> db.Student.update({_id:1},{$unset:{Location:"Network"}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

VIII. Finding Document based on search criteria suppressing few fields

```
db.Student.find({_id:1},{StudName:1,Grade:1,_id:0});
```

```
> db.Student.find({_id:1},{StudName:1,Grade:1,_id:0});
{ "StudName" : "Naman", "Grade" : "VII" }
```

To find those documents where the Grade is not set to 'VII'

```
db.Student.find({Grade:{$ne:'VII'}}).pretty();
```

```
> db.Student.find({Grade:{$ne:&apos;VII&apos;}}).pretty();
{ "_id" : 3, "StudName" : "abc", "Grade" : "VI" }
```


VIII. Finding Document based on search criteria suppressing few fields

```
db.Student.find({_id:1},{StudName:1,Grade:1,_id:0});
```

```
> db.Student.find({_id:1},{StudName:1,Grade:1,_id:0});
{ "StudName" : "Naman", "Grade" : "VII" }
```

To find those documents where the Grade is not set to 'VII'

```
db.Student.find({Grade:{$ne:'VII'}}).pretty();
```

```
> db.Student.find({Grade:{$ne:&apos;VII&apos;}}).pretty();
{ "_id" : 3, "StudName" : "abc", "Grade" : "VI" }
```

To find documents from the Students collection where the StudName ends with s.

```
db.Student.find({StudName:/s$/}).pretty();
```

```
> db.Student.find({StudName:/n$/}).pretty();
{
        "_id" : 1,
        "StudName" : "Naman",
        "Grade" : "VII",
        "Hobbies" : "InternetSurfing"
}
```

IX. to set a particular field value to NULL

```
db.Students.update({_id:3},{$set:{Location:null}})
```

```
> db.Student.update({_id:3},{$set:{Location:null}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

X. Count the number of documents in Student Collections

```
db.Students.count()
```

```
> db.Student.count();
3
```

Sort the document in Ascending order

db.Students.find().sort({StudName:1}).pretty();

```
> db.Student.find({Grade:"VII"}).limit(1);
{ "_id" : 1, "StudName" : "Naman", "Grade" : "VII", "Hobbies" : "InternetSurfing" }
```

Note:
for desending order : db.Students.find().sort({StudName:-1}).pretty();

to Skip the 1st two documents from the Students Collections

db.Students.find().skip(2).pretty()

XII. Create a collection by name “food” and add to each document add a “fruits” array

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

```
db.food.insert( { _id:1, fruits:['grapes','mango','apple'] } )
db.food.insert( { _id:2, fruits:['grapes','mango','cherry'] } )
db.food.insert( { _id:3, fruits:['banana','mango'] } )
```

```
> db.food.insert( { _id:1, fruits:['grapes','mango','apple'] } )
WriteResult({ "nInserted" : 1 })
> db.food.insert( { _id:2, fruits:['grapes','mango','cherry'] } )
WriteResult({ "nInserted" : 1 })
> db.food.insert( { _id:3, fruits:['banana','mango'] } )
WriteResult({ "nInserted" : 1 })
```

To find those documents from the "food" collection which has the "fruits array constitute of "grapes", "mango" and "apple".

db.food.find ({fruits: ['grapes','mango','apple'] }). pretty().

```
> db.food.find({fruits:['grapes','mango','apple']}).pretty();
```

```
{ "_id" : 1, "fruits" : [ "grapes", "mango", "apple" ] }
```

To find in "fruits" array having "mango" in the first index position.

db.food.find ({'fruits.1':'grapes'})

```
> db.food.find ( {'fruits.1':'mango'} )
{ "_id" : 1, "fruits" : [ "grapes", "mango", "apple" ] }
{ "_id" : 2, "fruits" : [ "grapes", "mango", "cherry" ] }
{ "_id" : 3, "fruits" : [ "banana", "mango" ] }
```

To find those documents from the "food" collection where the size of the array is two.

```
db.food.find ( {"fruits": {$size:2}} )

> db.food.find({"fruits":{$size:2}});
{ "_id" : 3, "fruits" : [ "banana", "mango" ] }
```

To find the document with a particular id and display the first two elements from the array "fruits"

```
db.food.find({_id:1},{"fruits":{$slice:2}})

> db.food.find({_id:1},{"fruits":{$slice:2}});
{ "_id" : 1, "fruits" : [ "grapes", "mango" ] }
```

To find all the documets from the food collection which have elements mango and grapes in the array "fruits"

```
db.food.find({fruits:{$all:["mango","grapes"]}})

> db.food.find({fruits:{$all:["mango","grapes"]}});
{ "_id" : 1, "fruits" : [ "grapes", "mango", "apple" ] }
{ "_id" : 2, "fruits" : [ "grapes", "mango", "cherry" ] }
```

update on Array:
using particular id replace the element present in the 1st index position of the fruits array with apple

```
db.food.update({_id:3},{$set:{'fruits.1':'apple'}})

> db.food.update({_id:3},{$set:{'fruits.1':'apple'}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

insert new key value pairs in the fruits array

```
db.food.update({_id:2},{$push:{price:{grapes:80,mango:200,cherry:100}}})

> db.food.update({_id:2},{$push:{price:{grapes:80,mango:200,cherry:100}}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

Note: perform query operations using - pop, addToSet, pullAll and pull

update on Array:
using particular id replace the element present in the 1st index position of the fruits array with apple

```
db.food.update({_id:3},{$set:{'fruits.1':'apple'}})

> db.food.update({_id:3},{$set:{'fruits.1':'apple'}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

insert new key value pairs in the fruits array

```
db.food.update({_id:2},{$push:{price:{grapes:80,mango:200,cherry:100}}})

> db.food.update({_id:2},{$push:{price:{grapes:80,mango:200,cherry:100}}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

Note: perform query operations using - pop, addToSet, pullAll and pull

XII. Aggregate Function :

Create a collection Customers with fields custID, AcctBal, AcctType.
Now group on "custID" and compute the sum of "AccBal".

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

db.Customers.aggregate ( {$group : { _id : "$custID",TotAccBal : {$sum:"$AccBal"} } } );

> db.Customers.aggregate({$group:{_id:"$custID",TotBal:{$sum:"$AcctBal"}}});
{ "_id" : 1234, "TotBal" : 100000 }
{ "_id" : 123, "TotBal" : 210000 }
```

match on AcctType:"S" then group on "CustID" and compute the sum of "AccBal".

db.Customers.aggregate ({$match:{AcctType:"S"}},{$group : { _id : "$custID",TotAccBal : {$sum:"$AccBal"} } });

match on AcctType:"S" then group on "CustID" and compute the sum of "AccBal" and total balance greater than 1200.

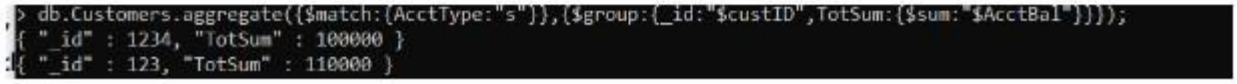
db.Customers.aggregate ({$match:{AcctType:"S"}},{$group : { _id : "$custID",TotAccBal : {$sum:"$AccBal"} } }, {$match:{TotAccBal:{$gt:1200}}});

match on AcctType:"S" then group on "CustID" and compute the sum of "AccBal".

db.Customers.aggregate ({$match:{AcctType:"S"}},{$group : { _id : "$custID",TotAccBal : {$sum:"$AccBal"} } });

match on AcctType:"S" then group on "CustID" and compute the sum of "AccBal" and total balance greater than 1200.

db.Customers.aggregate ({$match:{AcctType:"S"}},{$group : { _id : "$custID",TotAccBal : {$sum:"$AccBal"} } }, {$match:{TotAccBal:{$gt:1200}}});



Assignment:

Creation of Cursor:

Create Collection "Alphabets"
Insert Documents with fields "_id" and "alphabet"

use cursor to iterate through the "Alphabets" Collection.

```
> var cur = db.Alphabets.find();
> while(cur.hasNext()){
... print(tojson(cur.next()));}}
uncaught exception: SyntaxError: expected expression, got '}' :
@(shell):2:26
> while(cur.hasNext()){ print(tojson(cur.next()));}
{ "_id" : 0, "alphabet" : "a" }
{ "_id" : 1, "alphabet" : "b" }
{ "_id" : 2, "alphabet" : "c" }
>
```

MongoDB Lab 2: -

1) Using MongoDB

i) Create a database for Students and Create a Student Collection (_id,Name, USN, Semester, Dept_Name, CGPA, Hobbies(Set)). ii) Insert required documents to the collection.

iii) First Filter on “Dept_Name:CSE” and then group it on “Semester” and compute the Average CPGA for that semester and flter those documents where the “Avg_CPGA” is greater than 7.5.

iv) Command used to export MongoDB JSON documents from “Student” Collection into the “Students” database into a CSV fle “Output.txt”.

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

```
> db.Student.insert({_id:1,name:"ananya",USN:"1BM19CS095",Sem:6,Dept_Name:"CSE",CGPA:"8.1",Hobbies:"Badminton"});
WriteResult({ "nInserted" : 1 })
> db.Student.insert({_id:2,name:"bharath",USN:"1BM19CS002",Sem:6,Dept_Name:"CSE",CGPA:"8.3",Hobbies:"Swimming"});
WriteResult({ "nInserted" : 1 })
> db.Student.insert({_id:3,name:"chandana",USN:"1BM19CS006",Sem:6,Dept_Name:"CSE",CGPA:"7.1",Hobbies:"Cycling"});
WriteResult({ "nInserted" : 1 })
> db.Student.insert({_id:4,name:"hrithik",USN:"1BM19CS010",Sem:6,Dept_Name:"CSE",CGPA:"8.6",Hobbies:"Reading"});
WriteResult({ "nInserted" : 1 })
> db.Student.insert({_id:5,name:"kanika",USN:"1BM19CS090",Sem:6,Dept_Name:"CSE",CGPA:"9.2",Hobbies:"Cycling"});
WriteResult({ "nInserted" : 1 })
```

```
> db.Student.update({_id:1},{$set:{CGPA:9.0}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.Student.update({_id:2},{$set:{CGPA:9.1}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.Student.update({_id:3},{$set:{CGPA:8.1}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.Student.update({_id:4},{$set:{CGPA:6.5}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.Student.update({_id:5},{$set:{CGPA:8.6}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.students.aggregate({$match:{Dept_Name:"CSE"}},{$group:{_id:"$Sem",AvgCGPA:{$avg:"$CGPA"}}},{$match:{AvgCGPA:{$gt:7.5}}});
> db.Student.aggregate({$match:{Dept_Name:"CSE"}},{$group:{_id:"$Sem",AvgCGPA:{$avg:"$CGPA"}}},{$match:{AvgCGPA:{$gt:7.5}}});
{ "_id" : 6, "AvgCGPA" : 8.26 }
```

```
bmsce@bmsce-Precision-T1700:~$ mongoexport --host localhost --db nayana_db --collection Student --csv --out /home/bmsce/Desktop/output.txt
--fields "_id","Name","USN","Sem","Dept_Name","CGPA","Hobbies"
2022-04-20T15:13:53.933+0530    csv flag is deprecated; please use --type=csv instead
2022-04-20T15:13:53.935+0530    connected to: localhost
2022-04-20T15:13:53.935+0530    exported 5 records
```

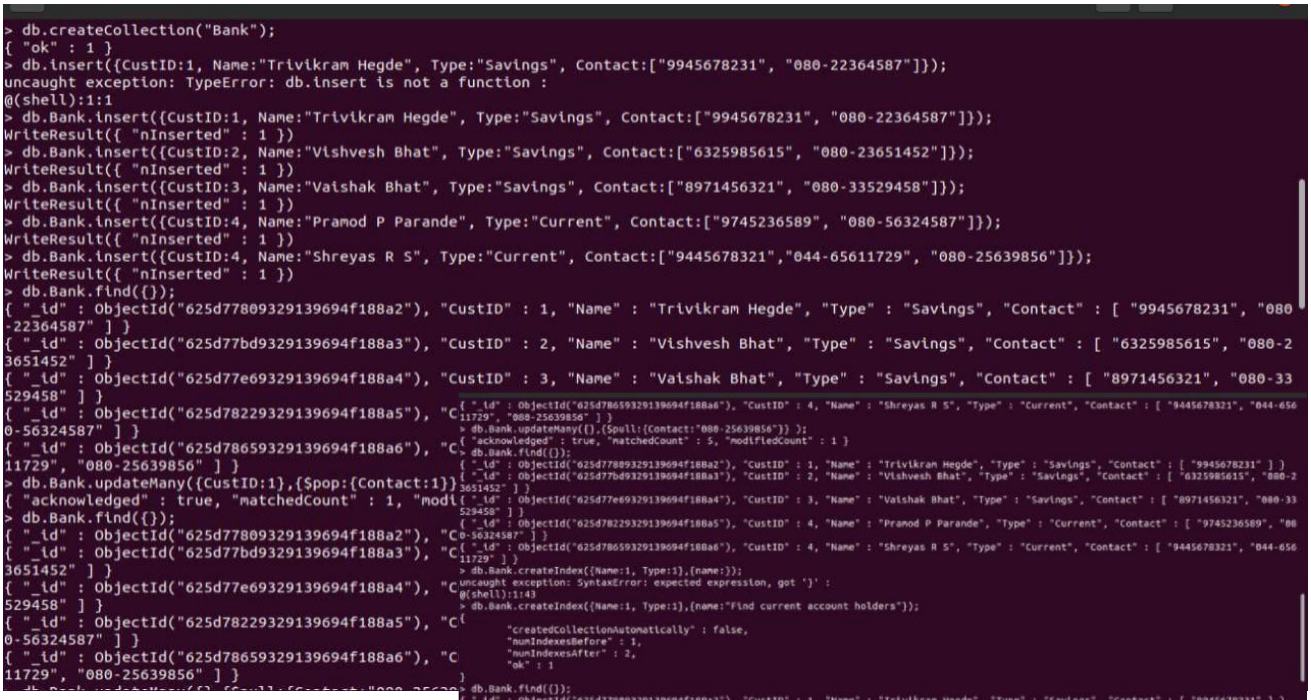
```
1 _id,Name,USN,Sem,Dept_Name,CGPA,Hobbies
2 1,,1BM19CS095,6,CSE,9,Badminton
3 2,,1BM19CS002,6,CSE,9.1,Swimming
4 3,,1BM19CS006,6,CSE,8.1,Cycling
5 4,,1BM19CS010,6,CSE,6.5,Reading
6 5,,1BM19CS090,6,CSE,8.6,Cycling
```

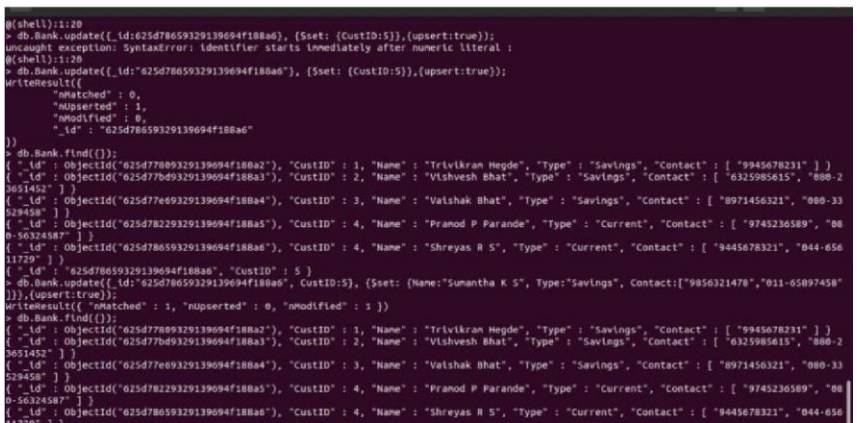
2)Create a mongodb collection Bank. Demonstrate the following by choosing felds of your choice.

1. Insert three documents
2. Use Arrays(Use Pull and Pop operation)
3. Use Index

4. Use Cursors

5. Updation





1) Using MongoDB,

i) Create a database for Faculty and Create a Faculty Collection(Faculty_id, Name, Designation ,Department, Age, Salary, Specialization(Set)). ii) Insert required documents to the collection.

iii) First Filter on “Dept_Name:MECH” and then group it on “Designation” and compute the Average Salary for that Designation and flter those documents where the “Avg_Sal” is greater than 650000. iv) Demonstrate usage of import and export commands

Write MongoDB queries for the following:

1)To display only the product name from all the documents of the product collection.

2)To display only the Product ID, ExpiryDate as well as the quantity from the document of the product collection where the _id column is 1.

```
}
> db.createCollection("faculty");
{ "ok" : 1 }
> db.faculty.insert({_id:1,name:"Dr. Balaraman Ravindran",designation:"Professor",department:"CSE",age:45,salary:100000,specialization:['pytho
n','mysql','sklearn', 'tensorflow']});
WriteResult({ "nInserted" : 1 })
> db.faculty.insert({_id:2,name:"Dr. Mahadev Ghorki",designation:"Assistant Professor",department:"CSE",age:35,salary:80000,specialization:['p
ython','numpy','sklearn', 'tensorflow', 'java']});
WriteResult({ "nInserted" : 1 })
> db.faculty.insert({_id:3,name:"Dr. Praveen Borade",designation:"Associate Professor",department:"ME",age:40,salary:75000,specialization:['au
tocad', 'aerodynamics', 'thermal physics']});
WriteResult({ "nInserted" : 1 })
> db.faculty.insert({_id:4,name:"Dr. Madhav Nayak",designation:"Assistant Professor",department:"ME",age:37,salary:95000,specialization:['auto
cad', 'flight-dynamics', 'Finite Element Analysis']});
WriteResult({ "nInserted" : 1 })
> db.faculty.aggregate ( {$match:{department:"ME"}}, {$group : {_id : "$designation", AverageSal :{$avg:"$salary"} } },  {$match:{AverageSal:{
$gt:50000}}});
{ "_id" : "Associate Professor", "AverageSal" : 75000 }
{ "_id" : "Assistant Professor", "AverageSal" : 95000 }
> db.createCollection("product");
{ "ok" : 1 }
> db.product.insert({pid:1,pname:"keyboard",mdate:2001,price:1800,quantity:2});
WriteResult({ "nInserted" : 1 })
> db.product.insert({pid:2,pname:"mouse",mdate:2005,price:1500,quantity:5});
WriteResult({ "nInserted" : 1 })
> db.product.insert({pid:3,pname:"monitor",mdate:2015,price:10000,quantity:9});
WriteResult({ "nInserted" : 1 })
> db.product.insert({pid:4,pname:"motherboard",mdate:2021,price:15000,quantity:4});
WriteResult({ "nInserted" : 1 })
> db.product.find({},{pname:1,_id:0})
{ "pname" : "keyboard" }
```

3)To fnd those documents where the price is not set to 15000.

4)To fnd those documents from the Product collection where the quantity is set to 9 and the product name is set to ‘monitor’.

5)To fnd documents from the Product collection where the Product name ends in ‘d’.

3)Create a mongodb collection Hospital. Demonstrate the following by choosing felds of choice.

1. Insert three documents
2. Use Arrays(Use Pull and Pop operation)
3. Use Index
4. Use Cursors
5. Updation

```
{ "pname" : "motherboard" }
> db.product.find({pid:1},{pid:1,_id:0,mdate:1,quantity:1});
{ "pid" : 1, "mdate" : 2001, "quantity" : 2 }
>  db.product.find({price:{$ne:15000}},{pname:1,_id:0});
{ "pname" : "keyboard" }
{ "pname" : "mouse" }
{ "pname" : "monitor" }
> db.product.find({$and:[{quantity:{$eq:9}},{pname:{$eq:"monitor"}}]},{pname:1,_id:0})
{ "pname" : "monitor" }
> db.product.find({pname:/d$/},{pname:1,quantity:1,_id:0})
{ "pname" : "keyboard", "quantity" : 2 }
{ "pname" : "motherboard", "quantity" : 4 }
> db.createCollection("hospital");
{ "ok" : 1 }
> db.hospital.insert({_id:1, Name: "Anshuman Agarwal", age:23, diseases:["fever", "diarrhoea", "wheezing", "gastritis"]});
WriteResult({ "nInserted" : 1 })
> db.hospital.insert({_id:2, Name: "Pinky Chaubey", age:35, diseases:["fever","nausea", "food infection", "indigestion", "kidney stones"]});
WriteResult({ "nInserted" : 1 })
> db.hospital.insert({_id:3, Name: "Amresh Chowpati", age:63, diseases:["hyperglycemia", "diabetes mellitus", "food poisoning", "cold"]});
WriteResult({ "nInserted" : 1 })
> db.hospital.updateMany({},{$pull:{diseases:"fever"}});
{ "acknowledged" : true, "matchedCount" : 3, "modifiedCount" : 2 }
> db.hospital.updateOne({_id:1},{$pop:{diseases:-1}});
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
> db.hospital.find({"diseases.2":"nausea"});
> db.hospital.find({"diseases.1":"nausea"});
> d.hospital.find({});
uncaught exception: ReferenceError: d is not defined :
@(shell):1:1
> db.hospital.find({});
{ "_id" : 1, "Name" : "Anshuman Agarwal", "age" : 23, "diseases" : [ "wheezing", "gastritis" ] }
{ "_id" : 2, "Name" : "Pinky Chaubey", "age" : 35, "diseases" : [ "nausea", "food infection", "indigestion", "kidney stones" ] }
{ "_id" : 3, "Name" : "Amresh Chowpati", "age" : 63, "diseases" : [ "hyperglycemia", "diabetes mellitus", "food poisoning", "cold" ] }
> db.hospital.find({"diseases.0":"nausea"});
{ "_id" : 2, "Name" : "Pinky Chaubey", "age" : 35, "diseases" : [ "nausea", "food infection", "indigestion", "kidney stones" ] }
> db.hospital.update({_id:3},{$set:{'diseases.1':'sarscov'}});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
```

Cassandra Lab Program 1: -

Perform the following DB operations using Cassandra.

1. Create a key space by name Employee

```
cqlsh> CREATE KEYSPACE employee WITH REPLICATION = {'class':'SimpleStrategy','replication_factor':1};
cqlsh> DESCRIBE KEYSPACES;

system_schema  system   system_distributed  system_traces
system_auth    samples  employee

cqlsh>
```

2. Create a column family by name Employee-Info with attributes Emp_Id Primary Key, Emp_Name, Designation, Date_of_Joining, Salary, Dept_Name

```
cqlsh:employee> CREATE TABLE EMPLOYEEINFO( EMPID INT, EMPNAME TEXT, DESIGNATION TEXT, DATEOFJOINING TIMESTAMP, SAL
ARY DOUBLE, DEPTNAME TEXT, PRIMARY KEY(EMPID,SALARY));
cqlsh:employee>
```

3. Insert the values into the table in batch

```
cqlsh:employee> BEGIN BATCH
           ... INSERT INTO EMPLOYEEINFO (EMPID, EMPNAME, DESIGNATION, DATEOFJOINING, SALARY, DEPTNAME)
           ... VALUES(1,'LOKESH','ASSISTANT MANAGER', '2005-04-6', 50000, 'MARKETING')
           ... INSERT INTO EMPLOYEEINFO (EMPID, EMPNAME, DESIGNATION, DATEOFJOINING, SALARY, DEPTNAME)
           ... VALUES(2,'DHEERAJ','ASSISTANT MANAGER', '2013-11-10', 30000, 'LOGISTICS')
           ... INSERT INTO EMPLOYEEINFO (EMPID, EMPNAME, DESIGNATION, DATEOFJOINING, SALARY, DEPTNAME)
           ... VALUES(3,'CHIRAG','ASSISTANT MANAGER', '2011-07-1', 115000, 'SALES')
           ... INSERT INTO EMPLOYEEINFO (EMPID, EMPNAME, DESIGNATION, DATEOFJOINING, SALARY, DEPTNAME)
           ... VALUES(4,'DHANUSH','ASSISTANT MANAGER', '2010-04-26', 75000, 'MARKETING')
           ...  INSERT INTO EMPLOYEEINFO (EMPID, EMPNAME, DESIGNATION, DATEOFJOINING, SALARY, DEPTNAME)
           ... VALUES(5,'ESHA','ASSISTANT MANAGER', '2010-04-26', 85000, 'TECHNICAL')
```

4. Update Employee name and Department of Emp-Id 121

```
cqlsh:employee> UPDATE EMPLOYEEINFO SET EMPNAME='HARRY', DEPTNAME='MANAGEMENT' WHERE EMPID=121
cqlsh:employee>  SELECT * FROM EMPLOYEEINFO;
```

5. Sort the details of Employee records based on salary

```
cqlsh:employee> select * from EMPLOYEEINFO where empid IN(1,2,3,4,5,6,7) ORDER BY salary DESC allow filtering;

 empid | salary   | dateofjoining                   | deptname  | designation       | empname
-------+----------+---------------------------------+-----------+-------------------+---------
     3 | 1.15e+05 | 2011-06-30 18:30:00.000000+0000 |     SALES | ASSISTANT MANAGER |  CHIRAG
     6 |    95000 | 2010-04-25 18:30:00.000000+0000 | TECHNICAL |           MANAGER |  FARHAN
     7 |    95000 | 2010-04-25 18:30:00.000000+0000 |        PR |           MANAGER |   JIMMY
     5 |    85000 | 2010-04-25 18:30:00.000000+0000 | TECHNICAL | ASSISTANT MANAGER |    ESHA
     4 |    75000 | 2010-04-25 18:30:00.000000+0000 | MARKETING | ASSISTANT MANAGER | DHANUSH
     1 |    50000 | 2005-04-05 18:30:00.000000+0000 | MARKETING | ASSISTANT MANAGER |  LOKESH
     2 |    30000 | 2013-11-09 18:30:00.000000+0000 | LOGISTICS | ASSISTANT MANAGER | DHEERAJ
```

6. Alter the schema of the table Employee_Info to add a column Projects which stores a set of Projects done by the corresponding Employee.

```
(7 rows)
cqlsh:employee> ALTER TABLE EMPLOYEEINFO ADD PROJECTS LIST<TEXT>;
cqlsh:employee> SELECT * FROM EMPLOYEEINFO;

 empid | salary    | dateofjoining                   | deptname    | designation       | empname | projects
-------+-----------+---------------------------------+-------------+-------------------+---------+----------
     5 |     85000 | 2010-04-25 18:30:00.000000+0000 |   TECHNICAL | ASSISTANT MANAGER |    ESHA |     null
     1 |     50000 | 2005-04-05 18:30:00.000000+0000 |   MARKETING | ASSISTANT MANAGER |  LOKESH |     null
     2 |     30000 | 2013-11-09 18:30:00.000000+0000 |   LOGISTICS | ASSISTANT MANAGER | DHEERAJ |     null
     4 |     75000 | 2010-04-25 18:30:00.000000+0000 |   MARKETING | ASSISTANT MANAGER | DHANUSH |     null
   121 |     99000 | 2010-04-25 18:30:00.000000+0000 | MANAGEMENT |  REGIONAL MANAGER |   HARRY |     null
     7 |     95000 | 2010-04-25 18:30:00.000000+0000 |          PR |           MANAGER |   JIMMY |     null
     6 |     95000 | 2010-04-25 18:30:00.000000+0000 |   TECHNICAL |           MANAGER |  FARHAN |     null
     3 | 1.15e+05 | 2011-06-30 18:30:00.000000+0000 |       SALES | ASSISTANT MANAGER |  CHIRAG |     null

(8 rows)
```

7. Update the altered table to add project names.

```
cqlsh:employee> UPDATE EMPLOYEEINFO SET PROJECTS=['FACEBOOK','SNAPCHAT'] WHERE EMPID=1 AND SALARY=50000;
cqlsh:employee> UPDATE EMPLOYEEINFO SET PROJECTS=['FACEBOOK','SNAPCHAT'] WHERE EMPID=7 AND SALARY=95000;
cqlsh:employee>  UPDATE EMPLOYEEINFO SET PROJECTS=['PINTEREST','INSTAGRAM'] WHERE EMPID=121 AND SALARY=99000;
cqlsh:employee> UPDATE EMPLOYEEINFO SET PROJECTS=['PINTEREST','INSTAGRAM'] WHERE EMPID=4 AND SALARY=75000;
cqlsh:employee> UPDATE EMPLOYEEINFO SET PROJECTS=['YOUTUBE','SPOTIFY'] WHERE EMPID=2 AND SALARY=30000;
cqlsh:employee> UPDATE EMPLOYEEINFO SET PROJECTS=['YOUTUBE','SPOTIFY'] WHERE EMPID=3 AND SALARY=115000;
cqlsh:employee> UPDATE EMPLOYEEINFO SET PROJECTS=['YOUTUBE','SPOTIFY'] WHERE EMPID=6 AND SALARY=95000;
cqlsh:employee> UPDATE EMPLOYEEINFO SET PROJECTS=['YOUTUBE','SPOTIFY'] WHERE EMPID=5 AND SALARY=85000;
cqlsh:employee> SELECT * FROM EMPLOYEEINFO;
```

8. Create a TTL of 15 seconds to display the values of Employees.

//BEFORE 15 seconds

```
cqlsh:employee> update EMPLOYEEINFO USING TTL 15  SET EMPNAME='LOKESH' where empid=1 AND salary=50000;
cqlsh:employee> SELECT * FROM EMPLOYEEINFO;

 empid | salary    | dateofjoining                   | deptname    | designation       | empname | projects
-------+-----------+---------------------------------+-------------+-------------------+---------+----------------------------

     5 |     85000 | 2010-04-25 18:30:00.000000+0000 |   TECHNICAL | ASSISTANT MANAGER |    ESHA |     ['YOUTUBE', 'SPOTIFY']
     1 |     50000 | 2005-04-05 18:30:00.000000+0000 |   MARKETING | ASSISTANT MANAGER |  LOKESH |   ['FACEBOOK', 'SNAPCHAT']
     2 |     30000 | 2013-11-09 18:30:00.000000+0000 |   LOGISTICS | ASSISTANT MANAGER | DHEERAJ |     ['YOUTUBE', 'SPOTIFY']
     4 |     75000 | 2010-04-25 18:30:00.000000+0000 |   MARKETING | ASSISTANT MANAGER | DHANUSH | ['PINTEREST', 'INSTAGRAM']
   121 |     99000 | 2010-04-25 18:30:00.000000+0000 | MANAGEMENT |  REGIONAL MANAGER |   HARRY | ['PINTEREST', 'INSTAGRAM']
     7 |     95000 | 2010-04-25 18:30:00.000000+0000 |          PR |           MANAGER |   JIMMY |   ['FACEBOOK', 'SNAPCHAT']
     6 |     95000 | 2010-04-25 18:30:00.000000+0000 |   TECHNICAL |           MANAGER |  FARHAN |     ['YOUTUBE', 'SPOTIFY']
     3 | 1.15e+05 | 2011-06-30 18:30:00.000000+0000 |       SALES | ASSISTANT MANAGER |  CHIRAG |     ['YOUTUBE', 'SPOTIFY']

(8 rows)
cqlsh:employee>
```

Cassandra - Lab 2

Create a key space by name Library

```
cqlsh> create keyspace Library WITH REPLICATION = {'class' : 'SimpleStrategy','replication_factor' :
1};
cqlsh> use Library;
```

Create a column family by name Library-Info with attributes Stud_Id Primary Key, Counter_value of type Counter,

```
cqlsh:library> create table Library_info(Stud_id int,Counter_value counter,Stud_Name varchar,Book_nam
e varchar,Book_id int,Date_of_issue date,primary key(Stud_id,Stud_name,Book_name,Book_id,Date_of_issu
e));
```

3. Insert the values into the table in batch

```
cqlsh:library> update library_info set Counter_value = Counter_value + 1 where Stud_id = 1 AND Stud_n
ame = 'naman' AND Book_name='abc' AND Book_id = 123 AND Date_of_issue = '2022-05-04';
```

Display the details of the table created and increase the value of the counter

```
cqlsh:library> update library_info set Counter_value = Counter_value + 1 where Stud_id = 1 AND Stud_n
ame = 'naman' AND Book_name='abc' AND Book_id = 123 AND Date_of_issue = '2022-05-04';
cqlsh:library> select * from Library_info;

 stud_id | stud_name | book_name | book_id | date_of_issue | counter_value
---------+-----------+-----------+---------+---------------+---------------
       1 |     naman |       abc |     123 |    2022-05-04 |             2
```

Write a query to show that a student with id 112 has taken a book “BDA” 2 times.

```
cqlsh:library> select counter_value as borrow_count from library_info where stud_id=1 AND book_id=123
;

 borrow_count
--------------
            2
```

xport the created column to a csv file

```
cqlsh:library> COPY library.library_info (Stud_id,Book_id,Counter_value,Stud_name,Book_name,Date_of_i
ssue) TO '/home/bmsce/CASSANDRA-NAMAN/data.csv' WITH HEADER = TRUE;
Using 11 child processes

Starting copy of library.library_info with columns [stud_id, book_id, counter_value, stud_name, book_
name, date_of_issue].
Processed: 1 rows; Rate:       6 rows/s; Avg. rate:       6 rows/s
1 rows exported to 1 files in 0.176 seconds.
```

Import a given csv dataset from local file system into Cassandra column family

```
cqlsh:library> COPY library.library_info (Stud_id,Book_id,Counter_value,Stud_name,Book_name,Date_of_i
ssue) FROM '/home/bmsce/CASSANDRA-NAMAN/data.csv' WITH HEADER = TRUE;
Using 11 child processes

Starting copy of library.library_info with columns [stud_id, book_id, counter_value, stud_name, book_
name, date_of_issue].
Processed: 1 rows; Rate:       2 rows/s; Avg. rate:       3 rows/s
1 rows imported from 1 files in 0.379 seconds (0 skipped).
```