

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



LAB REPORT

on

BIG DATA ANALYTICS (20CS6PEBDA)

Submitted by

Jathin SN (1BM19CS066)

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



B.M.S. COLLEGE OF ENGINEERING

(Autonomous Institution under VTU)

BENGALURU-560019

May-2022 to July-2022

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 **Jathin SN(1BM19CS229)**, who is bonafide 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.

Name of the Lab-In charge
Designation
Department of CSE
BMSCE, Bengaluru

ANTARA ROY CHOUDHURY
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Index Sheet

Sl. No.	Experiment Title	Page No.
1.	<u>Cassandra Lab Program 1:</u> - Create a Data set either structured/Semi-Structured/Unstructured from Twitter/Facebook etc. to perform various DB operations using Cassandra. (Use the Face Pager app to perform real-time streaming)	4
2.	<u>Cassandra Lab Program 2:</u> - Create a Data set either structured/Semi-Structured/Unstructured from Twitter/Facebook etc. to perform various DB operations using Cassandra. (Use the Face Pager app to perform real-time streaming)	7
3.	<u>MongoDB Lab Program 1 (CRUD Demonstration):</u> - Students should be classifying a dataset into one of the standard forms and apply suitable querying rules to obtain suitable results	10
4.	<u>MongoDB Lab Program 2 (CRUD Demonstration):</u> - Students should be classifying a dataset into one of the standard forms and apply suitable querying rules to obtain suitable results	21

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

Cassandra Lab Program 1: -

Perform the following DB operations using Cassandra.

1. Create a key space by name Employee

```
cqlsh> create keyspace Employee2 with replication = {'class':'SimpleStrategy','replication_factor':1};
cqlsh> describe Employee2;

CREATE KEYSPACE employee2 WITH replication = {'class': 'SimpleStrategy', 'replication_factor': '1'} AND durable_writes = true;
```

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> create table Employee2.Employee_Info(Emp_ID int Primary key,Emp_Name text,Designation text, Date_of_Joining timestamp,Salary double,Dept_Name text);
cqlsh> select * from Employee2.Employee_Info;
```

emp_id					
(0 rows)					

3. Insert the values into the table in batch

```
cqlsh> begin batch insert into Employee2.Employee_Info(emp_id,date_of_joining,dept_name,designation,emp_name,salary)values(2,'2020-03-07','Development','Manager','Tarun',1500000.0);insert into Employee2.Employee_Info(emp_id,date_of_joining,dept_name,designation,emp_name,salary)values(3,'2021-03-29','R&D','Web developer','Nithish',750000.0); apply batch;
cqlsh> select * from Employee2.Employee_Info;
```

emp_id					
2	2022-04-26 18:30:00.000000+0000	Development	Team lead	Prem Sai	1e+06
2	2020-03-06 18:30:00.000000+0000	Development	Manager	Tarun	1.5e+06
3	2021-03-28 18:30:00.000000+0000	R&D	Web developer	Nithish	7.5e+05

(3 rows)

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

```
cqlsh> update Employee2.Employee_Info set emp_name = 'Harsha',dept_name='Testing' where emp_id = 3;
cqlsh> select * from Employee2.Employee_Info;
```

emp_id					
1	2022-04-26 18:30:00.000000+0000	Deployment	Team lead	Prem Sai	1e+06
2	2020-03-06 18:30:00.000000+0000	Development	Manager	Tarun	1.5e+06
3	2021-03-28 18:30:00.000000+0000	Testing	Web developer	Harsha	7.5e+05

5. Sort the details of Employee records based on salary

```
cqlsh> create table Employee2.emp(Emp_Id int ,Salary double,primary key(Emp_Id,Salary));
cqlsh> begin batch
... insert into Employee2.emp(emp_Id,salary) values(1,1000000);
... insert into Employee2.emp(emp_Id,salary) values(2,1500000);
... insert into Employee2.emp(emp_Id,salary) values(3,700000);
... apply batch;
cqlsh> select * from Employee2.emp;
```

emp_id	salary
1	1e+06
2	1.5e+06

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.

```
cqlsh> alter table Employee2.Employee_Info add Projects set<text>;
cqlsh> select * from Employee2.Employee_Info ;
```

emp_id						
1	2022-04-26 18:30:00.000000+0000	Deployment	Team lead	Prem Sai	null	1e+06
2	2020-03-06 18:30:00.000000+0000	Development	Manager	Tarun	null	1.5e+06
3	2021-03-28 18:30:00.000000+0000	Testing	Web developer	Harsha	null	7.5e+05

Update the altered table to add project names.

8

```
cqlsh> update Employee2.Employee_Info set projects= projects+{'abc','xyz'} where emp_id=1;
cqlsh> select * from Employee2.Employee_Info ;
```

emp_id						
1	2022-04-26 18:30:00.000000+0000	Deployment	Team lead	Prem Sai	{'abc', 'xyz'}	1e+06
2	2020-03-06 18:30:00.000000+0000	Development	Manager	Tarun	null	1.5e+06
3	2021-03-28 18:30:00.000000+0000	Testing	Web developer	Harsha	null	7.5e+05

(3 rows)

```
cqlsh> update Employee2.Employee_Info set projects= projects+{'def','pqr'} where emp_id=2;
cqlsh> update Employee2.Employee_Info set projects= projects+{'gjd','ads'} where emp_id=3;
cqlsh> select * from Employee2.Employee_Info ;
```

emp_id						
1	2022-04-26 18:30:00.000000+0000	Deployment	Team lead	Prem Sai	{'abc', 'xyz'}	1e+06
2	2020-03-06 18:30:00.000000+0000	Development	Manager	Tarun	{'def', 'pqr'}	1.5e+06
3	2021-03-28 18:30:00.000000+0000	Testing	Web developer	Harsha	{'ads', 'gjd'}	7.5e+05

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

```
cqlsh> insert into Employee2.Employee_Info(emp_id,date_of_joining,dept_name,designation,emp_name,salary)values(10,'2020-02-27','Development','Intern','XYZ',150000.0) using TTL 15;
cqlsh> select * from Employee2.Employee_Info ;
```

emp_id						
10	2020-02-27 18:30:00.000000+0000	Development	Intern	XYZ	null	1.5e+05
1	2022-04-26 18:30:00.000000+0000	Deployment	Team lead	Prem Sai	{'abc', 'xyz'}	1e+06
2	2020-03-06 18:30:00.000000+0000	Development	Manager	Tarun	{'def', 'pqr'}	1.5e+06
3	2021-03-28 18:30:00.000000+0000	Testing	Web developer	Harsha	{'ads', 'gjd'}	7.5e+05

```
cqlsh> select * from Employee2.Employee_Info ;
```

emp_id						
1	2022-04-26 18:30:00.000000+0000	Deployment	Team lead	Prem Sai	{'abc', 'xyz'}	1e+06
2	2020-03-06 18:30:00.000000+0000	Development	Manager	Tarun	{'def', 'pqr'}	1.5e+06
3	2021-03-28 18:30:00.000000+0000	Testing	Web developer	Harsha	{'ads', 'gjd'}	7.5e+05

(3 rows)

Cassandra Lab Program 2: -

Perform the following DB operations using Cassandra.

1. Create a key space by name Library

```
cqlsh> create keyspace Library2 with replication = {'class':'SimpleStrategy','replication_factor':1};
cqlsh> describe Library2

CREATE KEYSPACE library2 WITH replication = {'class': 'SimpleStrategy', 'replication_factor': '1'} AND durable_
writes = true;
```

2. Create a column family by name Library-Info with attributes Stud_Id Primary Key, Counter_value of type Counter, Stud_Name, Book-Name, Book-Id, Date_of_issue

```
cqlsh:library2> create table library_info(Stud_id int,Counter_value counter,Stud_Name text,Book_N
ame text,Book_id text,Date_of_issue timestamp,Primary key(Stud_id,Stud_Name,Book_Name,Book_id,Dat
e_of_issue));
```

3. Insert the values into the table in batch

```
cqlsh:library2> update library_info set Counter_value=Counter_value+1 where Stud_Name='Roy' and S
tud_id=1 and Book_Name='BDA' and Book_id='101' and Date_of_issue='2022-04-22';
cqlsh:library2> update library_info set Counter_value=Counter_value+1 where Stud_Name='Tony' and
Stud_id=2 and Book_Name='CNS' and Book_id='102' and Date_of_issue='2022-04-16';
cqlsh:library2> update library_info set Counter_value=Counter_value+1 where Stud_Name='Prem' and
Stud_id=3 and Book_Name='BDA' and Book_id='103' and Date_of_issue='2022-02-15';
cqlsh:library2> update library_info set Counter_value=Counter_value+1 where Stud_Name='Prem' and
Stud_id=3 and Book_Name='BDA' and Book_id='106' and Date_of_issue='2022-02-20';
cqlsh:library2> select * from library_info
...
cqlsh:library2> select * from library_info;

 stud_id | stud_name | book_name | book_id | date_of_issue |
-----+-----+-----+-----+-----+
      1 |      Roy |      BDA |     101 | 2022-04-21 18:30:00.000000+0000 | 1
      2 |      Tony |      CNS |     102 | 2022-04-15 18:30:00.000000+0000 | 1
      3 |      Prem |      BDA |     103 | 2022-02-14 18:30:00.000000+0000 | 1
      3 |      Prem |      BDA |     106 | 2022-02-19 18:30:00.000000+0000 | 1

(4 rows)
```

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

```
cqlsh:library2> update library_info set Counter_value=Counter_value+1 where Stud_Name='Roy' and S
tud_id=1 and Book_Name='BDA' and Book_id='101' and Date_of_issue='2022-04-22';
cqlsh:library2> select * from library_info;
```

stud_id	stud_name	book_name	book_id	date_of_issue	counter_value
1	Roy	BDA	101	2022-04-21 18:30:00.000000+0000	2
2	Tony	CNS	102	2022-04-15 18:30:00.000000+0000	1
3	Prem	BDA	103	2022-02-14 18:30:00.000000+0000	1
3	Prem	BDA	106	2022-02-19 18:30:00.000000+0000	1

(4 rows)

```
cqlsh:library2> update library_info set Counter_value=Counter_value+1 where Stud_Name='Prem' and
Stud_id=3 and Book_Name='BDA' and Book_id='106' and Date_of_issue='2022-02-20';
cqlsh:library2> select * from library_info;
```

stud_id	stud_name	book_name	book_id	date_of_issue	counter_value
1	Roy	BDA	101	2022-04-21 18:30:00.000000+0000	2
2	Tony	CNS	102	2022-04-15 18:30:00.000000+0000	1
3	Prem	BDA	103	2022-02-14 18:30:00.000000+0000	1
3	Prem	BDA	106	2022-02-19 18:30:00.000000+0000	2

(4 rows)

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

```
cqlsh:library2> select * from library_info;
```

stud_id	stud_name	book_name	book_id	date_of_issue	counter_value
1	Roy	BDA	101	2022-04-21 18:30:00.000000+0000	2
2	Tony	CNS	102	2022-04-15 18:30:00.000000+0000	1
4	Arjun	BDA	201	2022-02-28 18:30:00.000000+0000	2
4	Arjun	BDA	201	2022-03-14 18:30:00.000000+0000	1
112	raj	BDA	521	2022-03-22 18:30:00.000000+0000	2
3	Prem	BDA	103	2022-02-14 18:30:00.000000+0000	1
3	Prem	BDA	106	2022-02-19 18:30:00.000000+0000	2

(7 rows)

```
cqlsh:library2> select * from library_info where Stud_id=112;
```

stud_id	stud_name	book_name	book_id	date_of_issue	counter_value
112	raj	BDA	521	2022-03-22 18:30:00.000000+0000	2

(1 rows)

6. Export the created column to a csv file

```
cqlsh:library2> copy library_info(stud_id,stud_name,book_name,book_id,date_of_issue,counter_value
) to '/home/bmsce/Desktop/bda/lib.csv';
Using 11 child processes

Starting copy of library2.library_info with columns [stud_id, stud_name, book_name, book_id, date
_of_issue, counter_value].
Processed: 7 rows; Rate: 45 rows/s; Avg. rate: 45 rows/s
7 rows exported to 1 files in 0.165 seconds.
cqlsh:library2> copy library_info1(stud_id,stud_name,book_name,book_id,date_of_issue,counter_valu
e) from '/home/bmsce/Desktop/bda/lib.csv';
```

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

```
cqlsh:library2> create table library_info1(Stud_id int,Counter_value counter,Stud_Name text,Book_Name text,Book_id text,Date_of_issue timestamp,Primary key(Stud_id,Stud_Name,Book_Name,Book_id,Date_of_issue));
cqlsh:library2> copy library_info1(stud_id,stud_name,book_name,book_id,date_of_issue,counter_value) from '/home/bmsce/Desktop/bda/lib.csv';
Using 11 child processes

Starting copy of library2.library_info1 with columns [stud_id, stud_name, book_name, book_id, date_of_issue, counter_value].
Processed: 7 rows; Rate:      13 rows/s; Avg. rate:      18 rows/s
7 rows imported from 1 files in 0.384 seconds (0 skipped).
cqlsh:library2> select * from library_info1;
```

stud_id	stud_name	book_name	book_id	date_of_issue	counter_value
1	Roy	BDA	101	2022-04-21 18:30:00.000000+0000	2
2	Tony	CNS	102	2022-04-15 18:30:00.000000+0000	1
4	Arjun	BDA	201	2022-02-28 18:30:00.000000+0000	2
4	Arjun	BDA	201	2022-03-14 18:30:00.000000+0000	1
112	raj	BDA	521	2022-03-22 18:30:00.000000+0000	2
3	Prem	BDA	103	2022-02-14 18:30:00.000000+0000	1
3	Prem	BDA	106	2022-02-19 18:30:00.000000+0000	2

(7 rows)

MongoDB Lab Program 1 (CRUD Demonstration): -

Execute the queries and upload a document with output.

I. CREATE DATABASE IN MONGODB.

use myDB;

db; (Confirm the existence of your database)

show dbs; (To list all databases)

```
Command Prompt - mongo
Microsoft Windows [Version 10.0.22000.675]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Admin>mongo
MongoDB shell version v5.0.9
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName=mongodb
Implicit session: session { "id" : UUID("484a3dd6-af99-4170-a440-b1c0987ab04e") }
MongoDB server version: 5.0.9
=====
Warning: the "mongo" shell has been superseded by "mongosh",
which delivers improved usability and compatibility. The "mongo" shell has been deprecated and will be removed in
an upcoming release.
For installation instructions, see
https://docs.mongodb.com/mongodb-shell/install/
=====
Welcome to the MongoDB shell.
For interactive help, type "help".
For more comprehensive documentation, see
https://docs.mongodb.com/
Questions? Try the MongoDB Developer Community Forums
https://community.mongodb.com
---
The server generated these startup warnings when booting:
  2022-06-03T06:17:24.092+05:30: Access control is not enabled for the database. Read and write access to data a
nd configuration is unrestricted
---
---
  Enable MongoDB's free cloud-based monitoring service, which will then receive and display
  metrics about your deployment (disk utilization, CPU, operation statistics, etc).

  The monitoring data will be available on a MongoDB website with a unique URL accessible to you
  and anyone you share the URL with. MongoDB may use this information to make product
  improvements and to suggest MongoDB products and deployment options to you.

  To enable free monitoring, run the following command: db.enableFreeMonitoring()
  To permanently disable this reminder, run the following command: db.disableFreeMonitoring()
---
> show dbs
admin    0.000GB
config  0.000GB
local    0.000GB
> use myDB;
switched to db myDB
> db;
myDB
> 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(...);

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

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

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:"Chess"}},{upsert:true});
```

```
local 0.000GB
> db.createCollection("Student");
{ "ok" : 1 }
> db.Student.drop();
true
> db.createCollection("Student");
{ "ok" : 1 }
> db.Student.insert({_id:1, StudName:"MichelleJacintha", Grade:"VII", Hobbies:"InternetSurfing"});
WriteResult({ "nInserted" : 1 })
> db.Student.insert({_id:1, StudName:"MichelleJacintha", Grade:"VII", Hobbies:"InternetSurfing"});
WriteResult({
  "nInserted" : 0,
  "writeError" : {
    "code" : 11000,
    "errmsg" : "E11000 duplicate key error collection: myDB.Student index: _id_ dup key: { _id: 1.0 }"
  }
})
> db.Student.updateelseinsert({_id:3, StudName:"AryanDavid", Grade:"VII"},{$set:{Hobbies:"Skating"}},{upsert:true});
uncaught exception: TypeError: db.Student.updateelseinsert is not a function :
@shell:1:1
> db.Student.update({_id:3, StudName:"AryanDavid", Grade:"VII"},{$set:{Hobbies:"Skating"}},{upsert:true});
WriteResult({ "nMatched" : 0, "nUpserted" : 1, "nModified" : 0, "_id" : 3 })
>
```

```
Command Prompt - mongo
> show collections
Student
> db.Student.find();
{ "_id" : 1, "StudName" : "MichelleJacintha", "Grade" : "VII", "Hobbies" : "InternetSurfing" }
{ "_id" : 3, "Grade" : "VII", "StudName" : "AryanDavid", "Hobbies" : "Skating" }
>
```

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:"AryanDavid"});
{ "_id" : 3, "Grade" : "VII", "StudName" : "AryanDavid", "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});
```

```
Command Prompt - mongo
> db.Student.find({}, {StudName:1, Grade:1, _id:0});
{ "StudName" : "MichelleJacintha", "Grade" : "VII" }
{ "Grade" : "VII", "StudName" : "AryanDavid" }
>
```

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

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

```
Command Prompt - mongo
> db.Student.find({Grade:{&#39;VII'&#39;}}).pretty();
{
  "_id" : 1,
  "StudName" : "MichelleJacintha",
  "Grade" : "VII",
  "Hobbies" : "InternetSurfing"
}
{
  "_id" : 3,
  "Grade" : "VII",
  "StudName" : "AryanDavid",
  "Hobbies" : "Skating"
}
>
```

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 ();
```

```
Command Prompt - mongo
> db.Student.find({Hobbies:{&#39;$in: [ 'Chess', 'Skating' ]&#39;}}).pretty();
{
  "_id" : 3,
  "Grade" : "VII",
  "StudName" : "AryanDavid",
  "Hobbies" : "Skating"
}
>
```

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

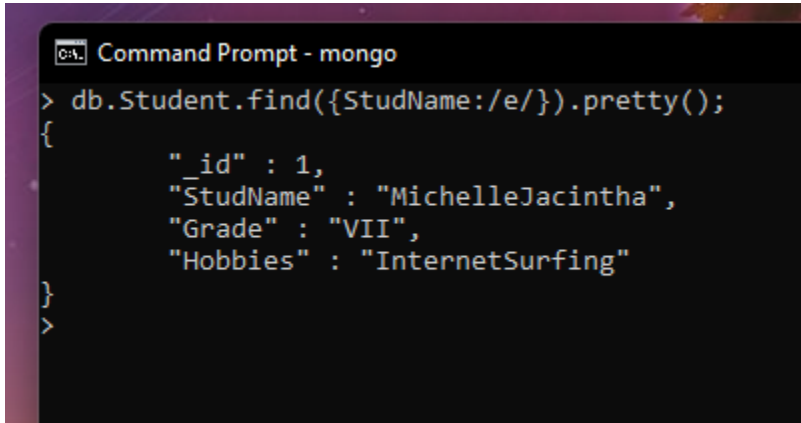
```
db.Student.find({StudName:/^M/}).pretty();
```

```
Command Prompt - mongo
> db.Student.find({StudName:/^M/}).pretty();
{
  "_id" : 1,
  "StudName" : "MichelleJacintha",
  "Grade" : "VII",
  "Hobbies" : "InternetSurfing"
}
>
```

F. To find documents from the Students collection where the StudName has an "e" in any

position.

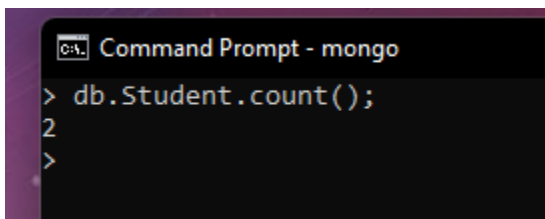
```
db.Student.find({StudName:/e/}).pretty();
```



```
Command Prompt - mongo
> db.Student.find({StudName:/e/}).pretty();
{
  "_id" : 1,
  "StudName" : "MichelleJacintha",
  "Grade" : "VII",
  "Hobbies" : "InternetSurfing"
}
```

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

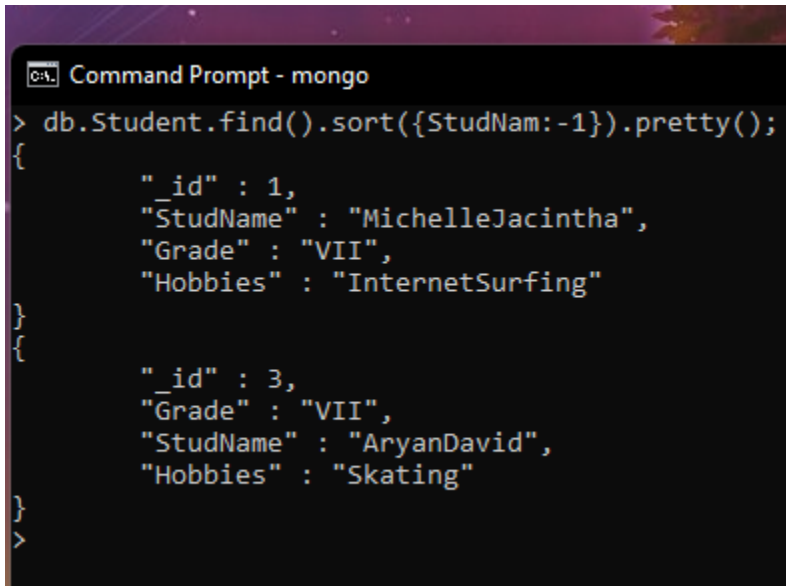
```
db.Student.count();
```



```
Command Prompt - mongo
> db.Student.count();
2
```

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

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

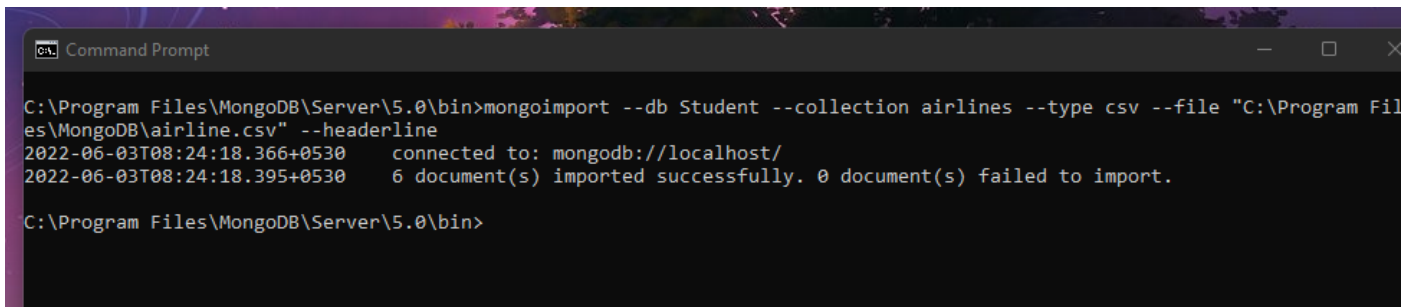


```
Command Prompt - mongo
> db.Student.find().sort({StudName:-1}).pretty();
{
  "_id" : 1,
  "StudName" : "MichelleJacintha",
  "Grade" : "VII",
  "Hobbies" : "InternetSurfing"
}
{
  "_id" : 3,
  "Grade" : "VII",
  "StudName" : "AryanDavid",
  "Hobbies" : "Skating"
}
```

III. 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
```

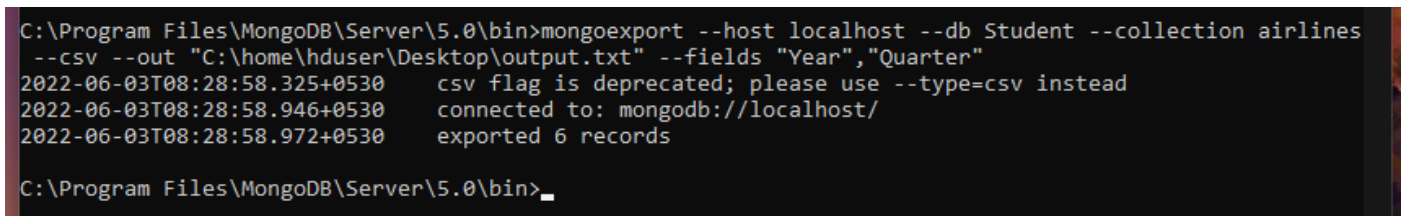


```
C:\Program Files\MongoDB\Server\5.0\bin>mongoimport --db Student --collection airlines --type csv --file "C:\Program Files\MongoDB\airline.csv" --headerline  
2022-06-03T08:24:18.366+0530    connected to: mongodb://localhost/  
2022-06-03T08:24:18.395+0530    6 document(s) imported successfully. 0 document(s) failed to import.  
C:\Program Files\MongoDB\Server\5.0\bin>
```

IV. 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"
```

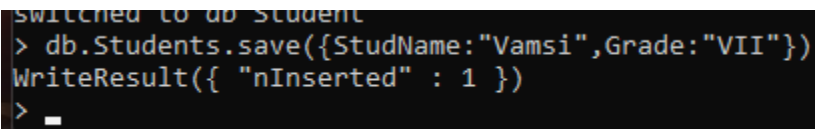


```
C:\Program Files\MongoDB\Server\5.0\bin>mongoexport --host localhost --db Student --collection airlines --csv --out "C:\home\hduser\Desktop\output.txt" --fields "Year","Quarter"  
2022-06-03T08:28:58.325+0530    csv flag is deprecated; please use --type=csv instead  
2022-06-03T08:28:58.946+0530    connected to: mongodb://localhost/  
2022-06-03T08:28:58.972+0530    exported 6 records  
C:\Program Files\MongoDB\Server\5.0\bin>
```

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 existing document.

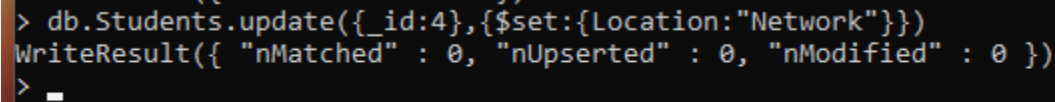
```
db.Students.save({StudName:"Vamsi", Grade:"VI"})
```



```
Switched to db Student  
> db.Students.save({StudName:"Vamsi",Grade:"VII"})  
WriteResult({ "nInserted" : 1 })  
> _
```

VI. Add a new field to existing Document:

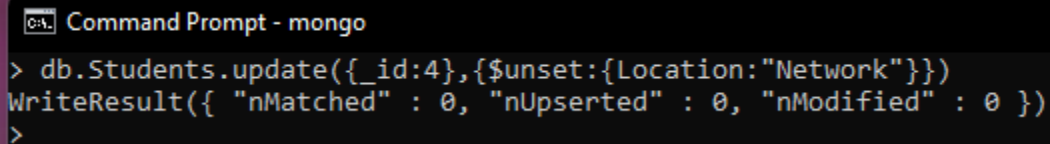
```
db.Students.update({_id:4},{ $set: {Location:"Network"}})
```



```
> db.Students.update({_id:4},{ $set: {Location:"Network"}})  
WriteResult({ "nMatched" : 0, "nUpserted" : 0, "nModified" : 0 })  
> _
```

VII. Remove the field in an existing Document

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



```
Command Prompt - mongo  
> db.Students.update({_id:4},{ $unset: {Location:"Network"}})  
WriteResult({ "nMatched" : 0, "nUpserted" : 0, "nModified" : 0 })  
>
```

VIII. Finding Document based on search criteria suppressing few fields

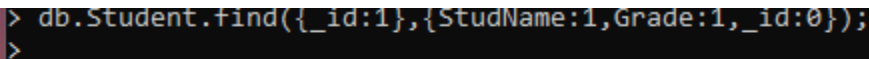
```
db.Student.find({_id:1},{StudName:1,Grade:1,_id:0});
```

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

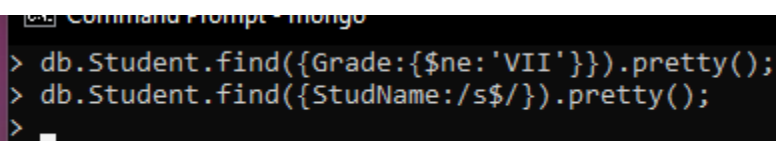
```
db.Student.find({Grade: {$ne: "&#39;VII&#39;"} }).pretty();
```

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

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



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



```
Command Prompt - mongo  
> db.Student.find({Grade: {$ne: 'VII'}}).pretty();  
> db.Student.find({StudName:/s$/}).pretty();  
> _
```

IX. to set a particular field value to NULL

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

X Count the number of documents in Student Collections

```
> db.Student.count()
0
>
```

XI. Count the number of documents in Student Collections with grade :VII

```
db.Students.count({Grade:"VII"})
```

retrieve first 3 documents

```
db.Students.find({Grade:"VII"}).limit(3).pretty();
```

Sort the document in Ascending order

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

Note:

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

to Skip the 1 st two documents from the Students Collections

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

```
> db.Students.find().sort({StudName:1}).pretty();
{
  "_id" : ObjectId("629979944de3211e43081306"),
  "StudName" : "Vamsi",
  "Grade" : "VII"
}
>
```

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

```
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'] } )
```



```

C:\> Command Prompt - mongo
> 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':'grapes'})
>

```

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 documents 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 1 st index position of the fruits array with apple

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

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:3},{ $set: {'fruits.1': 'apple'}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> 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.Customers.aggregate ( { $group : { _id : "$custID", TotAccBal : { $sum: "$AccBal" } } } );
```

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 } } } );
```

```
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.Customers.aggregate ( { $group : { _id : "$custID", TotAccBal : { $sum : "$AccBal" } } } );
> db.Customers.aggregate ( { $match : { AcctType : "S" } }, { $group : { _id : "$custID", TotAccBal :
... { $sum : "$AccBal" } } } );
uncaught exception: SyntaxError: illegal character :
@(shell):1:43
> db.Customers.aggregate ( { $match : { AcctType : "S" } }, { $group : { _id : "$custID", TotAccBal : { $sum : "$AccBal"
" } } } );
> db.Customers.aggregate ( { $match : { AcctType : "S" } }, { $group : { _id : "$custID", TotAccBal : { $sum : "$AccBa
l" } } }, { $match : { TotAccBal : { $gt : 1200 } } } );
>
```

MongoDB Lab Program 2 (CRUD Demonstration): -

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 filter 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 file "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 fields of your choice.

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

```

> db.createCollection("Bank");
{ "ok" : 1 }
> db.insert({CustID:1, Name:"Trivikram Hegde", Type:"Savings", Contact:["9945678231", "080-22364587"]});
uncaught exception: TypeError: db.insert is not a function :
@ (shell):1:11
> db.Bank.insert({CustID:1, Name:"Trivikram Hegde", Type:"Savings", Contact:["9945678231", "080-22364587"]});
WriteResult({ "nInserted" : 1 })
> db.Bank.insert({CustID:2, Name:"Vishvesh Bhat", Type:"Savings", Contact:["6325985615", "080-23651452"]});
WriteResult({ "nInserted" : 1 })
> db.Bank.insert({CustID:3, Name:"Vaishak Bhat", Type:"Savings", Contact:["8971456321", "080-33529458"]});
WriteResult({ "nInserted" : 1 })
> db.Bank.insert({CustID:4, Name:"Pramod P Parande", Type:"Current", Contact:["9745236589", "080-56324587"]});
WriteResult({ "nInserted" : 1 })
> db.Bank.insert({CustID:4, Name:"Shreyas R S", Type:"Current", Contact:["9445678321", "044-65611729", "080-25639856"]});
WriteResult({ "nInserted" : 1 })
> db.Bank.find({});
{ "_id" : ObjectId("625d77809329139694f188a2"), "CustID" : 1, "Name" : "Trivikram Hegde", "Type" : "Savings", "Contact" : [ "9945678231", "080-22364587" ] }
{ "_id" : ObjectId("625d77bd9329139694f188a3"), "CustID" : 2, "Name" : "Vishvesh Bhat", "Type" : "Savings", "Contact" : [ "6325985615", "080-23651452" ] }
{ "_id" : ObjectId("625d77e69329139694f188a4"), "CustID" : 3, "Name" : "Vaishak Bhat", "Type" : "Savings", "Contact" : [ "8971456321", "080-33529458" ] }
{ "_id" : ObjectId("625d78229329139694f188a5"), "CustID" : 4, "Name" : "Pramod P Parande", "Type" : "Current", "Contact" : [ "9745236589", "080-56324587" ] }
{ "_id" : ObjectId("625d78659329139694f188a6"), "CustID" : 4, "Name" : "Shreyas R S", "Type" : "Current", "Contact" : [ "9445678321", "044-65611729", "080-25639856" ] }
> db.Bank.updateMany({CustID:1},{ $pop:{Contact:1} });
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }
> db.Bank.find({});
{ "_id" : ObjectId("625d77809329139694f188a2"), "CustID" : 1, "Name" : "Trivikram Hegde", "Type" : "Savings", "Contact" : [ "9945678231" ] }
{ "_id" : ObjectId("625d77bd9329139694f188a3"), "CustID" : 2, "Name" : "Vishvesh Bhat", "Type" : "Savings", "Contact" : [ "6325985615", "080-23651452" ] }
{ "_id" : ObjectId("625d77e69329139694f188a4"), "CustID" : 3, "Name" : "Vaishak Bhat", "Type" : "Savings", "Contact" : [ "8971456321", "080-33529458" ] }
{ "_id" : ObjectId("625d78229329139694f188a5"), "CustID" : 4, "Name" : "Pramod P Parande", "Type" : "Current", "Contact" : [ "9745236589", "080-56324587" ] }
{ "_id" : ObjectId("625d78659329139694f188a6"), "CustID" : 4, "Name" : "Shreyas R S", "Type" : "Current", "Contact" : [ "9445678321", "044-65611729", "080-25639856" ] }

```



```

{ "_id" : ObjectId("625d78659329139694f188a6"), "CustID" : 4, "Name" : "Shreyas R S", "Type" : "Current", "Contact" : [ "9445678321", "044-65611729", "080-25639856" ] }
> db.Bank.updateMany({},{$pull:{Contact:"080-25639856"}});
{ "acknowledged" : true, "matchedCount" : 5, "modifiedCount" : 1 }
> db.Bank.find({});
{ "_id" : ObjectId("625d77809329139694f188a2"), "CustID" : 1, "Name" : "Trivikram Hegde", "Type" : "Savings", "Contact" : [ "9945678231" ] }
{ "_id" : ObjectId("625d77bd9329139694f188a3"), "CustID" : 2, "Name" : "Vishvesh Bhat", "Type" : "Savings", "Contact" : [ "6325985615", "080-23651452" ] }
{ "_id" : ObjectId("625d77e69329139694f188a4"), "CustID" : 3, "Name" : "Vaishak Bhat", "Type" : "Savings", "Contact" : [ "8971456321", "080-33529458" ] }
{ "_id" : ObjectId("625d78229329139694f188a5"), "CustID" : 4, "Name" : "Pranod P Parande", "Type" : "Current", "Contact" : [ "9745236589", "080-56324587" ] }
{ "_id" : ObjectId("625d78659329139694f188a6"), "CustID" : 4, "Name" : "Shreyas R S", "Type" : "Current", "Contact" : [ "9445678321", "044-65611729" ] }
> db.Bank.createIndex({Name:1, Type:1},{name:''});
uncaught exception: SyntaxError: expected expression, got '' :
@ (shell):1:43
> db.Bank.createIndex({Name:1, Type:1},{name:"Find current account holders"});
{
  "createdCollectionAutomatically" : false,
  "numIndexesBefore" : 1,
  "numIndexesAfter" : 2,
  "ok" : 1
}
> db.Bank.find({});
{ "_id" : ObjectId("625d77809329139694f188a2"), "CustID" : 1, "Name" : "Trivikram Hegde", "Type" : "Savings", "Contact" : [ "9945678231" ] }
{ "_id" : ObjectId("625d77bd9329139694f188a3"), "CustID" : 2, "Name" : "Vishvesh Bhat", "Type" : "Savings", "Contact" : [ "6325985615", "080-23651452" ] }
{ "_id" : ObjectId("625d77e69329139694f188a4"), "CustID" : 3, "Name" : "Vaishak Bhat", "Type" : "Savings", "Contact" : [ "8971456321", "080-33529458" ] }
{ "_id" : ObjectId("625d78229329139694f188a5"), "CustID" : 4, "Name" : "Pranod P Parande", "Type" : "Current", "Contact" : [ "9745236589", "080-56324587" ] }
{ "_id" : ObjectId("625d78659329139694f188a6"), "CustID" : 4, "Name" : "Shreyas R S", "Type" : "Current", "Contact" : [ "9445678321", "044-65611729" ] }
> db.Bank.getIndexes()
[
  {
    "v" : 2,
    "key" : {
      "Name" : 1,
      "Type" : 1
    },
    "name" : "Find current account holders"
  }
]

```

```

@ (shell):1:20
> db.Bank.update({_id:625d78659329139694f188a6}, {$set: {CustID:5}}, {upsert:true});
uncaught exception: Identifier starts immediately after numeric literal :
@ (shell):1:20
> db.Bank.update({_id:"625d78659329139694f188a6"}, {$set: {CustID:5}}, {upsert:true});
WriteResult({
  "nMatched" : 0,
  "nUpserted" : 1,
  "nModified" : 0,
  "_id" : "625d78659329139694f188a6"
})
> db.Bank.find({});
{ "_id" : ObjectId("625d77809329139694f188a2"), "CustID" : 1, "Name" : "Trivikram Hegde", "Type" : "Savings", "Contact" : [ "9945678231" ] }
{ "_id" : ObjectId("625d77bd9329139694f188a3"), "CustID" : 2, "Name" : "Vishvesh Bhat", "Type" : "Savings", "Contact" : [ "6325985615", "080-23651452" ] }
{ "_id" : ObjectId("625d77e69329139694f188a4"), "CustID" : 3, "Name" : "Vaishak Bhat", "Type" : "Savings", "Contact" : [ "8971456321", "080-33529458" ] }
{ "_id" : ObjectId("625d78229329139694f188a5"), "CustID" : 4, "Name" : "Pranod P Parande", "Type" : "Current", "Contact" : [ "9745236589", "080-56324587" ] }
{ "_id" : ObjectId("625d78659329139694f188a6"), "CustID" : 4, "Name" : "Shreyas R S", "Type" : "Current", "Contact" : [ "9445678321", "044-65611729" ] }
{ "_id" : "625d78659329139694f188a6", "CustID" : 5 }
> db.Bank.update({_id:"625d78659329139694f188a6", CustID:5}, {$set: {Name:"Sumantha K S", Type:"Savings", Contact:["9856321478", "011-65897458"]}}, {upsert:true});
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.Bank.find({});
{ "_id" : ObjectId("625d77809329139694f188a2"), "CustID" : 1, "Name" : "Trivikram Hegde", "Type" : "Savings", "Contact" : [ "9945678231" ] }
{ "_id" : ObjectId("625d77bd9329139694f188a3"), "CustID" : 2, "Name" : "Vishvesh Bhat", "Type" : "Savings", "Contact" : [ "6325985615", "080-23651452" ] }
{ "_id" : ObjectId("625d77e69329139694f188a4"), "CustID" : 3, "Name" : "Vaishak Bhat", "Type" : "Savings", "Contact" : [ "8971456321", "080-33529458" ] }
{ "_id" : ObjectId("625d78229329139694f188a5"), "CustID" : 4, "Name" : "Pranod P Parande", "Type" : "Current", "Contact" : [ "9745236589", "080-56324587" ] }
{ "_id" : ObjectId("625d78659329139694f188a6"), "CustID" : 4, "Name" : "Shreyas R S", "Type" : "Current", "Contact" : [ "9445678321", "044-65611729" ] }
{ "_id" : "625d78659329139694f188a6", "CustID" : 5, "Contact" : [ "9856321478", "011-65897458" ], "Name" : "Sumantha K S", "Type" : "Savings" }
>

```

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 filter 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.
- 3)To find those documents where the price is not set to 15000.
- 4)To find those documents from the Product collection where the quantity is set to 9 and the product name is set to ‘monitor’.
- 5)To find documents from the Product collection where the Product name ends in ‘d’.

```
> db.createCollection("faculty");
{ "ok" : 1 }
> db.faculty.insert({_id:1,name:"Dr. Balaraman Ravindran",designation:"Professor",department:"CSE",age:45,salary:100000,specialization:['python','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:['python','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:['autocad','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:['autocad','flight-dynamics','Finite Element Analysis']});
WriteResult({ "nInserted" : 1 })
> db.faculty.aggregate ( {$match:{department:"ME"}}, {$group : {_id : "$designation", AverageSal : {$avg:"$salary"} } }, {$match:{AverageSal:{$gt:50000}}});
{ "_id" : "Associate Professor", "AverageSal" : 75000 }
{ "_id" : "Assistant Professor", "AverageSal" : 95000 }
> db.createCollection("product");
{ "ok" : 1 }
> db.product.insert({pid:1,pname:"keyboard",mdate:2001,price:1800,quantity:2});
WriteResult({ "nInserted" : 1 })
> db.product.insert({pid:2,pname:"mouse",mdate:2005,price:1500,quantity:5});
WriteResult({ "nInserted" : 1 })
> db.product.insert({pid:3,pname:"monitor",mdate:2015,price:10000,quantity:9});
WriteResult({ "nInserted" : 1 })
> db.product.insert({pid:4,pname:"motherboard",mdate:2021,price:15000,quantity:4});
WriteResult({ "nInserted" : 1 })
> db.product.find({},{pname:1,_id:0})
{ "pname" : "keyboard" }
{ "pname" : "mouse" }
{ "pname" : "monitor" }
{ "pname" : "motherboard" }
> db.product.find({pid:1},{pid:1,_id:0,mdate:1,quantity:1});
{ "pid" : 1, "mdate" : 2001, "quantity" : 2 }
> db.product.find({price:{$ne:15000}},{pname:1,_id:0});
{ "pname" : "keyboard" }
{ "pname" : "mouse" }
```

3) Create a MongoDB collection Hospital. Demonstrate the following by choosing fields 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 })
> █
```