

# VISVESVARAYA TECHNOLOGICAL UNIVERSITY

“JnanaSangama”, Belgaum -590014, Karnataka.



**LAB REPORT on**

## **BIG DATA ANALYTICS (20CS6PEBDA)**

*Submitted by*

**Khushil M Sindhwad (1BM19CS072)**

*in partial fulfilment 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**” carried out by **Khushil M Sindhwad(1BM19CS072)**, 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 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	MongoDB- CRUD Demonstration	
2	Cassandra Lab Program 1: - Student Database	
3	Cassandra Lab Program 2: - Library Database	

## 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

## LAB 1:

### I.CREATE DATABASE IN MONGODB.

> use khushilDB

switched to db khushilDB

db;

khushilDB

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(...);**

**{ "ok" : 1 }**

- 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"});**

**WriteResult({ "nInserted" : 1 })**

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});
```

```
WriteResult({ "nMatched" : 0, "nUpserted" : 1, "nModified" : 0, "_id" : 3 })
```

## 5.FIND METHOD

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

```
db.Student.find({StudName:"AryanDavid"});  
({cond..},{columnns.. column:1, columnname:0} )
```

```
{ "_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});
```

```
{ "StudName" : "MichelleJacintha", "Grade" : "VII" }  
{ "Grade" : "VII", "StudName" : "AryanDavid" }
```

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

```
db.Student.find({Grade:{Seq:'VII'}}).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 ();
```

```
{
  "_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();
```

```
{
```

```
"_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();
```

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

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

```
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();
```

```
{  
  "_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
```

### 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"
```

### 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.Student.save({StudName:"Vamsi", Grade:"VI"})
```



```
WriteResult({ "nInserted" : 1 })
```

#### **VI. Add a new field to existing Document:**

```
db.Student.update({_id:ObjectId("625695cc7d129fb98b44c8a1")},  
{ $set:{Location:"Network"}})
```

```
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
```

#### **VII. Remove the field in an existing Document**

```
db.Student.update({_id:ObjectId("625695cc7d129fb98b44c8a1"  
"})),  
{ $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});
```

```
{ "StudName" : "MichelleJacintha", "Grade" : "VII" }
```

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

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

```
{
```

```
  "_id" : ObjectId("625695cc7d129fb98b44c8a1"),
```

```
  "StudName" : "Vamsi",
```

```
"Grade" : "VI"
```

```
}
```

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

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

```
{
```

```
  "_id" : 1,
```

```
  "StudName" : "MichelleJacintha",
```

```
  "Grade" : "VII",
```

```
  "Hobbies" : "InternetSurfing"
```

```
}
```

**IX. to set a particular field value to 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.Student.count()
```

```
3
```

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

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

## 2 retrieve first 3 documents

```
db.Student.find({Grade:"VII"}).limit(1).pretty();
```

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

## Sort the document in Ascending order

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

```
{
  "_id" : 3,
  "Grade" : "VII",
  "StudName" : "AryanDavid",
  "Hobbies" : "Skating",
  "Location" : null
}
{
  "_id" : 1,
  "StudName" : "MichelleJacintha",
  "Grade" : "VII",
  "Hobbies" : "InternetSurfing"
}
{
  "_id" : ObjectId("625695cc7d129fb98b44c8a1"),
  "StudName" : "Vamsi",
  "Grade" : "VI"
```

```
}
```

**Note: for descending order :**

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

**to Skip the 1<sup>st</sup> two documents from the Students Collections**

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

```
{  
  "_id" : ObjectId("625695cc7d129fb98b44c8a1"),  
  "StudName" : "Vamsi",  
  "Grade" : "VI"  
}
```

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'] } )
```

```
{ "_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 which has the “fruits array” constitute of “grapes”, “mango” and “apple”.**

```
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'} )
```

**To find those documents from the “food” collection where the size of the array is two.**

```
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}})
```

```
{ "_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”]}})
```

```
{ "_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<sup>st</sup> index position of the fruits array with 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}}})
```

```
{ "_id" : 1, "fruits" : [ "grapes", "mango", "apple" ] }  
{ "_id" : 2, "fruits" : [ "grapes", "mango", "cherry" ], "price" : [ {  
  "grapes" : 80, "mango" : 200, "cherry" : 100 } ] }  
{ "_id" : 3, "fruits" : [ "banana", "apple" ] }
```

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

## LAB 2:

Perform the following DB operations using Cassandra.

1. Create a key space by name Employee

```
create keyspace "Employee" with replication =  
{'class':'SimpleStrategy','replication_factor':1}; cqlsh>  
use Employee;
```

2. Create a column family by name Employee-Info with attributes  
Emp\_Id Primary Key, Emp\_Name, Designation, Date\_of\_Joining,  
Salary, Dept\_Name

```
create table Employee_Info(Emp_id int PRIMARY KEY,Emp_name  
text,Date_of_Joining timestamp,Salary float,Dept_Name text) ;
```

3. Insert the values into the table in batch

```
cqlsh:employee> begin batch  
... insert into  
Employee_Info(Emp_id,Emp_name,Date_of_Joining,Salary,Dept_N  
ame) values(1,'Khushil','2021-04-23',50000,'CSE')  
... insert into  
Employee_Info(Emp_id,Emp_name,Date_of_Joining,Salary,Dept_N  
ame) values(2,'Tarun','2020-06-21',10000,'ISE')  
... insert into  
Employee_Info(Emp_id,Emp_name,Date_of_Joining,Salary,Dept_N  
ame) values(3,'Suresh','2011-02-12',30000,'ECE')  
... insert into  
Employee_Info(Emp_id,Emp_name,Date_of_Joining,Salary,Dept_N ame)  
values(4,'Yuresh','2015-09-02',90000,'EEE')
```

... insert into  
Employee\_Info(Emp\_id,Emp\_name,Date\_of\_Joining,Salary,Dept\_Name) values(5,'Dharmesh','2016-01-09',70000,'CSE')  
... apply batch;

```
cqlsh> create keyspace Employee with replication = {'class':'SimpleStrategy', 'replication_factor':1};
cqlsh> use Employee
... ;
cqlsh:employee> create table Employee_Info(Emp_id int PRIMARY KEY,Emp_name text,Date_of_Joining timestamp,Salary float,Dept_Name text);
cqlsh:employee> begin batch
... insert into Employee_Info(Emp_id,Emp_name,Date_of_Joining,Salary,Dept_Name) values(1,'Nithin','2021-04-23',50000,'CSE')
... insert into Employee_Info(Emp_id,Emp_name,Date_of_Joining,Salary,Dept_Name) values(2,'Tarun','2020-06-21',10000,'ISE')
... insert into Employee_Info(Emp_id,Emp_name,Date_of_Joining,Salary,Dept_Name) values(3,'Suresh','2011-02-12',30000,'ECE')
... insert into Employee_Info(Emp_id,Emp_name,Date_of_Joining,Salary,Dept_Name) values(4,'Yuresh','2015-09-02',90000,'EEE')
... insert into Employee_Info(Emp_id,Emp_name,Date_of_Joining,Salary,Dept_Name) values(5,'Dharmesh','2016-01-09',70000,'CSE')
... apply batch;
cqlsh:employee> select * from Employee_info;
```

emp_id	date_of_joining	dept_name	emp_name	salary
5	2016-01-09 00:00:00.000000+0000	CSE	Dharmesh	70000
1	2021-04-23 00:00:00.000000+0000	CSE	Nithin	50000
2	2020-06-21 00:00:00.000000+0000	ISE	Tarun	10000
4	2015-09-02 00:00:00.000000+0000	EEE	Yuresh	90000
3	2011-02-12 00:00:00.000000+0000	ECE	Suresh	30000

- Update Employee name and Department of Emp-Id 1 update employee\_info set Dept\_Name='Mech',emp\_name='Sreekar' where emp\_id=1;
- cqlsh:employee> select \* from employee\_info;

```
cqlsh:employee> select * from employee_info;
```

emp_id	date_of_joining	dept_name	emp_name	salary
5	2016-01-09 00:00:00.000000+0000	CSE	Dharmesh	70000
1	2021-04-23 00:00:00.000000+0000	Mech	Sreekar	50000
2	2020-06-21 00:00:00.000000+0000	ISE	Tarun	10000
4	2015-09-02 00:00:00.000000+0000	EEE	Yuresh	90000
3	2011-02-12 00:00:00.000000+0000	ECE	Suresh	30000

(5 rows)



## 6. Sort the details of Employee records based on salary

```
(0 rows)
cqlsh:employee> begin batch
... insert into Employee_information(Emp_id,Emp_name,Date_of_Joi
ning,Salary,Dept_Name) values(1,'Nithin','2021-04-23',50000,'CSE')
... insert into Employee_information(Emp_id,Emp_name,Date_of_Joi
ning,Salary,Dept_Name) values(2,'Tarun','2020-06-21',10000,'ISE')
... insert into Employee_information(Emp_id,Emp_name,Date_of_Joi
ning,Salary,Dept_Name) values(3,'Suresh','2011-02-12',30000,'ECE')
... apply batch;
cqlsh:employee> select * from Employee_information;
```

emp_id	salary	date_of_joining	dept_name	emp_name
1	50000	2021-04-23 00:00:00.000000+0000	CSE	Nithin
2	10000	2020-06-21 00:00:00.000000+0000	ISE	Tarun
3	30000	2011-02-12 00:00:00.000000+0000	ECE	Suresh

```
(3 rows)
cqlsh:employee> describe Employee_information;

CREATE TABLE employee.employee_information (
  emp_id int,
  salary float,
  date_of_joining timestamp,
  dept_name text,
  emp_name text,
  PRIMARY KEY (emp_id, salary)
) WITH CLUSTERING ORDER BY (salary ASC)
```

cqlsh:employee> select \* from Employee\_information where emp\_id in (1,2,3) order by Salary;

```
cqlsh:employee> paging off
Disabled Query paging.
cqlsh:employee> select * from Employee_information where emp_id in (1,2,3) o
rder by Salary;
```

emp_id	salary	date_of_joining	dept_name	emp_name
2	10000	2020-06-21 00:00:00.000000+0000	ISE	Tarun
3	30000	2011-02-12 00:00:00.000000+0000	ECE	Suresh
1	50000	2021-04-23 00:00:00.000000+0000	CSE	Nithin

```
(3 rows)
```

## 7. Alter the schema of the table Employee\_Info to add a column Projects which stores a set of Projects done by the corresponding Employee.

cqlsh:employee> alter table employee\_info add projects set<text>;

8. Update the altered table to add project names.

```
cqlsh:employee> update employee_info set  
projects=projects+{'project1','project2','project3'} where emp_id=1;
```

```
cqlsh:employee> select * from employee_info;
```

emp_id	date_of_joining	dept_name	emp_name	projects	salary
5	2016-01-09 00:00:00.000000+0000	CSE	Dharmesh	null	70000
1	2021-04-23 00:00:00.000000+0000	Mech	Sreekar	{'project1', 'project2', 'project3'}	50000
2	2020-06-21 00:00:00.000000+0000	ISE	Tarun	null	10000
4	2015-09-02 00:00:00.000000+0000	EEE	Yuresh	null	90000
3	2011-02-12 00:00:00.000000+0000	ECE	Suresh	null	30000

(5 rows)

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

```
cqlsh:employee> begin batch  
... insert into Employee_Info(Emp_id,Emp_name,Date_of_Joining,Salary,Dept_Name) values(6,'Rahul','2021-05-03',10000,'ISE') USING TTL 15;  
... apply batch;  
cqlsh:employee> select * from employee_info;
```

emp_id	date_of_joining	dept_name	emp_name	projects	salary
5	2016-01-09 00:00:00.000000+0000	CSE	Dharmesh	null	70000
1	2021-04-23 00:00:00.000000+0000	Mech	Sreekar	{'project1', 'project2', 'project3'}	50000
2	2020-06-21 00:00:00.000000+0000	ISE	Tarun	{'project4', 'project5'}	10000
4	2015-09-02 00:00:00.000000+0000	EEE	Yuresh	null	90000
6	2021-05-03 00:00:00.000000+0000	ISE	Rahul	null	10000
3	2011-02-12 00:00:00.000000+0000	ECE	Suresh	null	30000

(6 rows)

```
cqlsh:employee> select * from employee_info;
```

emp_id	date_of_joining	dept_name	emp_name	projects	salary
5	2016-01-09 00:00:00.000000+0000	CSE	Dharmesh	null	70000
1	2021-04-23 00:00:00.000000+0000	Mech	Sreekar	{'project1', 'project2', 'project3'}	50000
2	2020-06-21 00:00:00.000000+0000	ISE	Tarun	{'project4', 'project5'}	10000
4	2015-09-02 00:00:00.000000+0000	EEE	Yuresh	null	90000
3	2011-02-12 00:00:00.000000+0000	ECE	Suresh	null	30000

(5 rows)

### LAB 3:

1. Create a key space by name Library

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

2. 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_name varchar, Book_Id Int, Date_of_Issue date, primary key(Stud_Id, Stud_Name, Book_name, Book_Id, Date_of_Issue));
```

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_Name = 'naman' AND Book_name='abc' AND Book_Id = 123 AND Date_of_Issue = '2022-05-04';
```

4. 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_Name = '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

5. 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             |
+-----+
```

## 6. Export 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_issue) 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.
```

## 7. 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_issue) 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).
```