**BDA LAB REPORT**

**Aditi A Firangi**

**6A, 1BM19CS006**

**CSE Department**

**Pallavi G B**

**Demonstration of MongoDB commands**

I. CREATE DATABASE IN MONGODB.

use myDB;

Confirm the existence of your database

db;

To list all databases

show dbs;

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”);

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

db.Student.drop();

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

db.Student.insert({\_id:1,StudName:”MichelleJacintha”,Grade:”VII”,Hobbies:”InternetS

urfing”});

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:”Skatin

g”}},{upsert:true});

5. FIND METHOD

A. To search for documents from the “Students” collection based on certain search

criteria.

db.Student.find({StudName:”Aryan David”});

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});

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

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

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 ();

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

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

F. To find documents from the Students collection where the StudNamehas an “e” in any

position.

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

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

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

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

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

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

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();

Sort the document in Ascending order

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

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 exisiting document.

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

VI. Add a new field to existing Document:

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

VII. Remove the field in an existing Document

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

IX. to set a particular field value to NULL

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

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

db.Students.count({Grade:”VII”})

XI. Retrieve first 3 documents

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

XII. Tto Skip the 1 st two documents from the Students Collections

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

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’] } );

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();

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}} )

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}})

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”]}})

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}}})

**Demonstration of DB operations using Cassandra**

A

1.Create a keyspace by name Employee

create keyspace Emp1 with replication={

'class':'SimpleStrategy','replication\_factor':1};

use Emp1;

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 emp\_info(

empid int PRIMARY KEY,

empName text,

designation text,

date\_of\_joining timestamp,

deptName text,

salary int );

3. Insert the values into the table in batch

BEGIN BATCH

INSERT INTO emp\_info(empid, empName, designation, date\_of\_joining, deptName, salary) VALUES(111, 'Anne', 'Manager', '2019-05-22', 'Sales', 100000)

INSERT INTO emp\_info(empid, empName, designation, date\_of\_joining, deptName, salary) VALUES(123, 'Ben', 'Tester', '2021-07-21', 'IT', 90000)

INSERT INTO emp\_info(empid, empName, designation, date\_of\_joining, deptName, salary) VALUES(121, 'Cate', 'Developer', '2021-11-19', 'IT', 110000)

APPLY BATCH;

select \* from emp\_info;

empid | date\_of\_joining | deptname | designation | empname | salary

-------+---------------------------------+----------+-------------+---------+--------

123 | 2021-07-20 18:30:00.000000+0000 | IT | Tester | Ben | 90000

111 | 2019-05-21 18:30:00.000000+0000 | Sales | Manager | Anne | 100000

121 | 2021-11-18 18:30:00.000000+0000 | IT | Developer | Cate | 110000

(3 rows)

4. Update Employee name and Department of Emp-Id 121  
update emp\_info set empName='David', deptName='HR' where empid=121;

select \* from emp\_info;

empid | date\_of\_joining | deptname | designation | empname | salary

-------+---------------------------------+----------+-------------+---------+--------

123 | 2021-07-20 18:30:00.000000+0000 | IT | Tester | Ben | 90000

111 | 2019-05-21 18:30:00.000000+0000 | Sales | Manager | Anne | 100000

121 | 2021-11-18 18:30:00.000000+0000 | HR | Developer | David | 110000

(3 rows)

5. Sort the details of Employee records based on salary  
  
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.

ALTER TABLE emp\_info ADD Projects set<text>;

select \* from emp\_info;

empid | date\_of\_joining | deptname | designation | empname | projects | salary

-------+---------------------------------+----------+-------------+---------+----------+--------

123 | 2021-07-20 18:30:00.000000+0000 | IT | Tester | Ben | null | 90000

111 | 2019-05-21 18:30:00.000000+0000 | Sales | Manager | Anne | null | 100000

121 | 2021-11-18 18:30:00.000000+0000 | IT | Developer | Cate | null | 110000

(3 rows)

7. Update the altered table to add project names.

UPDATE emp\_info SET Projects={'Social Media Marketing'} WHERE empid=111;

UPDATE emp\_info SET Projects={'Security','IoT'} WHERE empid=121;

UPDATE emp\_info SET Projects={'Machine Learning'} WHERE empid=123;

select \* from emp\_info;

empid | date\_of\_joining | deptname | designation | empname | projects | salary

-------+---------------------------------+----------+-------------+---------+----------------------------+--------

123 | 2021-07-20 18:30:00.000000+0000 | IT | Tester | Ben | {'Machine Learning'} | 90000

111 | 2019-05-21 18:30:00.000000+0000 | Sales | Manager | Anne | {'Social Media Marketing'} | 100000

121 | 2021-11-18 18:30:00.000000+0000 | IT | Developer | Cate | {'IoT', 'Security'} | 110000

(3 rows)

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

BEGIN BATCH

INSERT INTO emp\_info(empid, empName, designation, date\_Of\_joining, deptName, Projects, salary) VALUES(124,'Joe','Intern','2022-03-20', 'IT', {'Security'}, 50000) USING TTL 15

APPLY BATCH;

select \* from emp\_info;

empid | date\_of\_joining | deptname | designation | empname | projects | salary

-------+---------------------------------+----------+-------------+---------+----------------------------+--------

123 | 2021-07-20 18:30:00.000000+0000 | IT | Tester | Ben | {'Machine Learning'} | 90000

111 | 2019-05-21 18:30:00.000000+0000 | Sales | Manager | Anne | {'Social Media Marketing'} | 100000

121 | 2021-11-18 18:30:00.000000+0000 | IT | Developer | Cate | {'IoT', 'Security'} | 110000

124 | 2022-03-19 18:30:00.000000+0000 | IT | Intern | Joe | {'Security'} | 50000

B

1.Create a keyspace by name Library

 cqlsh> create keyspace library\_space WITH REPLICATION={'class':'SimpleStrategy','replication\_factor':2};

cqlsh> use library\_space;

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:library\_space> create table library\_info(stud\_id int,counter\_value counter,stud\_name text,book\_name text,book\_id int,date\_of\_issue timestamp,PRIMARY KEY(stud\_id,stud\_name,book\_name,book\_id,date\_of\_issue));

3. Insert the values into the table in batch

cqlsh:library\_space> update library\_info set counter\_value=counter\_value+1 where stud\_id=1 and stud\_name='abc' and book\_name='book1' and book\_id=11 and date\_of\_issue='2022-01-30';

cqlsh:library\_space> update library\_info set counter\_value=counter\_value+1 where stud\_id=2 and stud\_name='def' and book\_name='book2' and book\_id=12 and date\_of\_issue='2022-03-30';

cqlsh:library\_space> update library\_info set counter\_value=counter\_value+1 where stud\_id=3 and stud\_name='ghi' and book\_name='book3' and book\_id=13 and date\_of\_issue='2022-05-30';

cqlsh:library\_space> update library\_info set counter\_value=counter\_value+1 where stud\_id=4 and stud\_name='jkl' and book\_name='book4' and book\_id=14 and date\_of\_issue='2022-07-30';

cqlsh:library\_space> update library\_info set counter\_value=counter\_value+1 where stud\_id=5 and stud\_name='mno' and book\_name='book5' and book\_id=15 and date\_of\_issue='2022-09-30';

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

cqlsh:library\_space> select \* from library\_info;

stud\_id | stud\_name | book\_name | book\_id | date\_of\_issue | counter\_value

---------+-----------+-----------+---------+---------------------------------+---------------

5 | mno | book5 | 15 | 2022-09-29 18:30:00.000000+0000 | 1

1 | abc | book1 | 11 | 2022-01-29 18:30:00.000000+0000 | 1

2 | def | book2 | 12 | 2022-03-29 18:30:00.000000+0000 | 1

4 | jkl | book4 | 14 | 2022-07-29 18:30:00.000000+0000 | 1

3 | ghi | book3 | 13 | 2022-05-29 18:30:00.000000+0000 | 1

(5 rows)

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

cqlsh:library\_space> update library\_info set counter\_value=counter\_value+1 where stud\_id=5 and stud\_name='mno' and book\_name='book5' and book\_id=15 and date\_of\_issue='2022-09-30';

cqlsh:library\_space> select \* from library\_info;

stud\_id | stud\_name | book\_name | book\_id | date\_of\_issue | counter\_value

---------+-----------+-----------+---------+---------------------------------+---------------

5 | mno | book5 | 15 | 2022-09-29 18:30:00.000000+0000 | 2

1 | abc | book1 | 11 | 2022-01-29 18:30:00.000000+0000 | 1

2 | def | book2 | 12 | 2022-03-29 18:30:00.000000+0000 | 1

4 | jkl | book4 | 14 | 2022-07-29 18:30:00.000000+0000 | 1

3 | ghi | book3 | 13 | 2022-05-29 18:30:00.000000+0000 | 1

(5 rows)

              
6. Export the created column to a csv file

cqlsh:library\_space> copy library\_info(stud\_id,stud\_name,book\_name,book\_id,date\_of\_issue,counter\_value) to '/home/bmscecse/Desktop/bda.csv';

Using 11 child processes

Starting copy of library\_space.library\_info with columns [stud\_id, stud\_name, book\_name, book\_id, date\_of\_issue, counter\_value].

Processed: 5 rows; Rate: 45 rows/s; Avg. rate: 45 rows/s

5 rows exported to 1 files in 0.121 seconds.

              
7. Import a given csv dataset from local file system into Cassandra  
column family  
cqlsh:library\_space> create table library\_info\_copy(stud\_id int,counter\_value counter,stud\_name text,book\_name text,book\_id int,date\_of\_issue timestamp,PRIMARY KEY(stud\_id,stud\_name,book\_name,book\_id,date\_of\_issue));

cqlsh:library\_space> copy library\_info\_copy(stud\_id,stud\_name,book\_name,book\_id,date\_of\_issue,counter\_value) from '/home/bmscecse/Desktop/new.csv';

Using 11 child processes

Starting copy of library\_space.library\_info\_copy with columns [stud\_id, stud\_name, book\_name, book\_id, date\_of\_issue, counter\_value].

Processed: 5 rows; Rate: 8 rows/s; Avg. rate: 12 rows/s

5 rows imported from 1 files in 0.406 seconds (0 skipped).

cqlsh:library\_space> select \* from library\_info where counter\_value=2 allow filtering;

stud\_id | stud\_name | book\_name | book\_id | date\_of\_issue | counter\_value

---------+-----------+-----------+---------+---------------------------------+---------------

5 | mno | book5 | 15 | 2022-09-29 18:30:00.000000+0000 | 2

**Execution of HDFS Commands for interaction with Hadoop Environment. (Minimum 9 commands to be executed)**

1. mkdir

Hadoop HDFS mkdir Command Usage

mkdir

Hadoop HDFS mkdir Command Example

hdfs dfs -mkdir /abc

Hadoop HDFS mkdir Command Description

This HDFS command takes path URI’s as an argument and creates directories.

2. ls

Hadoop HDFS ls Command Usage

ls

Hadoop HDFS ls Command Example

hadoop fs -ls /

Hadoop HDFS ls Commnad Description

This Hadoop HDFS ls command displays a list of the contents of a directory specified by path provided by the user, showing the names, permissions, owner, size and modification date for each entry.

3. put

Hadoop HDFS put Command Usage

put

Hadoop HDFS put Command Example

hdfs dfs -put /home/hduser/Desktop/Welcome.txt /abc/WC.txt

Hadoop

HDFS put Command Description

This hadoop basic command copies the file or directory from the local file system to the destination within the DFS.

4. copyFromLocal

Hadoop HDFS copyFromLocal Command Usage

copyFromLocal

Hadoop HDFS copyFromLocal Command Example

hdfs dfs -put /home/hduser/Desktop/Welcome.txt /abc/WC.txt

Hadoop HDFS copyFromLocal Command Description

This hadoop shell command is similar to put command, but the source is restricted to a local file reference.

Display the contents of the file WC2.txt

hdfs dfs -cat /abc/WC2.txt

5. get

Hadoop HDFS get Command Usage

get [-crc]

i.Hadoop HDFS get Command Example

hdfs dfs -get /abc/WC.txt /home/hduser/Downloads/WWC.txt

This HDFS fs command copies the file or directory in HDFS identified by the source to the local file system path identified by local destination.

ii.Hadoop HDFS get Command Example

hdfs dfs -getmerge /abc/WC.txt /abc/WC2.txt /home/hduser/Desktop/Merge.txt

This HDFS basic command retrieves all files that match to the source path entered by the user in HDFS, and creates a copy of them to one single, merged file in the local file system identified by local destination.

iii. Hadoop HDFS get Command Example

hadoop fs -getfacl /abc/

This Apache Hadoop command shows the Access Control Lists (ACLs) of files and directories.

6. copyToLocal

Hadoop HDFS copyToLocal Command Usage

copyToLocal

Hadoop HDFS copyToLocal Command Example

hdfs dfs -copyToLocal /abc/WC.txt /home/hduser/Desktop

Similar to get command, only the difference is that in this the destination is restricted to a local file reference.

7. cat

Hadoop HDFS cat Command Usage

cat

Hadoop HDFS cat Command Example

hdfs dfs -cat /abc/WC.txt

This Hadoop fs shell command displays the contents of the filename on console or stdout.

8. mv

Hadoop HDFS mv Command Usage

mv

Hadoop HDFS mv Command Example

hadoop fs -mv /abc /FFF

hadoop fs -ls /FFF

This basic HDFS command moves the file or directory indicated by the source to destination, within HDFS.

9. cp

Hadoop HDFS cp Command Usage

cp

Hadoop HDFS cp Command Example

hadoop fs -cp /CSE/ /LLL

hadoop fs -ls /LLL

The cp command copies a file from one directory to another directory within the HDFS.