04/03/2025

DATE: PAGE:

Lab1

1. Create database in mongoDB.

- use myDB
- db
- show dbs

2. Create, read, update, delete operations

- db.createCollection("Student");
- db.Student.drop();
- db.createCollection("Students");
- db.Students.insert({_id:1, StudName:"Michelle Jacintha", grade:"VII", Hobbies:"internet surfing"});
- db.Students.update({_id:3, StudName:"AryanDavid", grade:"} {$set:{Hobbies:"skating"}}, {upsert:true});
- db.Students.find({StudName:"AryanDavid"});
- db.Students.find({},{StudName:1, grade:1, _id:0});
- db.students.find({grade:{$eq:'VII'}}).pretty();
- db.students.find({Hobbies:{$in:['chess','skating']}}).pretty();
- db.Students.find({StudName:/^M/}).pretty();
- db.Students.find({StudName:/e/}).pretty();
- db.Students.count();
- db.Students.find().sort({StudName:-1}).pretty();

5. Save method

db.Students.save({StudName:"Vamsi", grade:"V"})

6. Add a new field to existing document.

-> db.Students.update({_id:4}, {$set:{location:"Network"}})

7. Remove the field in an existing document.

-> db.Students.update({_id:4}, {$unset:{location:"Network"}}).

8. Finding document based on search criteria

-> db.Students.find({_id:1}, {StudName:1, grade:1, _id:0});
-> db.Students.find({grade:{$ne:'VIII'}}).pretty();
-> db.Students.find({StudName:/S$/}).pretty();

9. To set a particular field value to NULL

-> db.Students.update({_id:3}, {$set:{location:null}})

10. count the number of documents in student collections.

-> db.Students.count();

-> db.Students.count({grade:"VI"});
-> db.Students.find({grade:"VII"}).limit(3).pretty();
-> db.Students.find().sort({StudName:1}).pretty();
-> db.Students.find().sort({StudName:-1}).pretty();
-> db.Students.find().skip(2).pretty()

Aggregate function :

DATE: PAGE:

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

- db.customerss.aggregate([{$match:{Acctype:"S"}},{$group:
{_id:"$custID",TotAccBal:{$sum:"$AccBal"}}});

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

DATE: PAGE:

Lab-02

Mongodb Lab Exercise.

Perform the following DB operations using MongoDB.

a) Create a collection by name customers with the following attributes.

cust_id, ACC_Bal, ACC_Type.

```
db.createcollection("customers");
```

b) Insert at least 5 values into the table.

```
db.Customers.insertMany([
{cust_id:1, ACC_Bal:500, Acc_Type:"z"},
{cust_id:2, ACC_Bal:1100, ACC_Type:"A"},
{cust_id:3, ACC_Bal:2000, ACC_Type:"z"},
{cust_id:4, ACC_Bal:900, Acc_Type:"B"},
{cust_id:5, ACC_Bal:1300, Acc_Type:"z"}])
```

c) write a query to display those records whose total account balance is greater than 1200 of account type 'z' for each customer_id.

```
db.customers.find({ACC_Bal:{$gt:1200}, ACC_Type:"z"})
```

d) Determine Minimum and maximum account balance for each customer.

DATE: PAGE:

```
db.customers.aggregate([
  $group:{
    _id: "$cust_id",
    Min_Balance:{$min: "$ACC_Bal"},
    Max_Balance:{$max: "$ACC_Bal"}}}])
```

2. You are developing an e-commerce platform where users can browse and purchase products. Each product has a unique identifier, a name, a category, a price, and available quantity. Additionally, users can add products to their cart and place orders. Design a query to

```
db.createCollection("Products")
db.Products.insertMany([
{_id:1, name:"Laptop", category:"Electronics", price:1000,
  quantity:5},
{_id:2, name:"Phone", category:"Electronics", price:500,
  quantity:10},
{_id:3, name:"T-shirt", category:"clothing", price:30,
  quantity:20},
{_id:4, name:"Headphones", category:"Electronics", price:150,
quantity:15},
{_id:5, name:"shoes", category:"Footwear", price:80,
  quantity:12}])

db.createCollection("users")
db.users.insertMany([
{_id:"789ghi", name:"John Doe", cart:[{product_id:1,
quantity:1}, {product_id:3, quantity:2}]},
{_id:"456xyz", name:"Alice smith", cart:[{product_id:2,
quantity:1}, {product_id:4, quantity:1}]}])
```

DATE: PAGE:

db.createCollection("orders")

db.Orders.insertMany([

{_id:"123abc", user_id:"789ghi", products:[{product_id:1, quantity:1}, {product_id:3, quantity:2}], total_price:1060},

{_id:"456def", user_id:"456xyz", products:[{product_id:2, quantity:1}, {product_id:4, quantity:1}], total_price:650}])

a) Retrieve All products:

db.Products.find({})

b) Retrieve products in a specific category (e.g., electronics)

db.Products.find({category:"Electronics"})

c) Retrieve products with quantity greater than 0.

db.Products.find({quantity:{$gt:0}})

d) Retrieve products sorted by price in Ascending order.

db.Products.find().sort({price:1})

e) Retrieve products with price less than or equal to $100.

db.Products.find({price:{$lte:100}})

f) Retrieve products added to a user's cart (id with "789ghi").

db.Users.find({_id:"789ghi"},{cart:1}).

g) Retrieve orders placed by a user (with "123abc").

db.Orders.find({_id:"123abc"})

DATE: PAGE:

5) Retrieve total price of orders placed by a user (with "123abc").

db.orders.aggregate([{ $match: { _id: "123abc" } }, { $group: { _id: "$_id", TotalPrice: { $sum: "$total_price" } } }])

Additional Aggregation

1. Calculate total Number of Products in each category.

db.Products.aggregate([{ $group: { _id: "$category", TotalProducts: { $sum: 1 } } }])

2. Calculate total price of products in each category.

db.Products.aggregate([{ $group: { _id: "$category", TotalPrice: { $sum: "$price" } } }])

3. Find Average Price of products.

db.Products.aggregate([{ $group: { _id: null, AvgPrice: { $avg: "$price" } } }])

4. Find products with quantity less than 10.

db.Products.find({ quantity: { $lt: 10 } })

5. Sort products by price in descending order.

db.Products.find().sort({ price: -1 })

6. Calculate total price of orders placed by each user.

db.orders.aggregate([{ $group: { _id: "$user_id", Total_price: { $sum: "$total_price" } } }])

7. Find users with the highest total price of orders.

```
db.Orders.aggregate([
{$group:{_id:"$user_id", TotalPrice:{$sum:"$total_price"}}},
{$sort:{TotalPrice:-1}}, {$limit:1}])
```

Find Average total price of others.

```
db.Orders.aggregate([
{$group:{_id:null, AvgTotalPrice:{$avg:"$total_price"}}}
])
```

11/3/25

1/04/23 DATE: PAGE:

working with cassandra

- CREATE KEYSPACE Students WITH REPLICATION = {'class':'SimpleStrategy', 'replication_factor':1};

- DESCRIBE KEYSPACES;

- select * from system_schema.keyspaces;

- use Students

- create table 'Students_info (Roll_No int PRIMARY KEY, Studname text, Date_of_joining timestamp, last_exam_Percent double)

- describe tables;
- describe system_schema.tables.

- Begin Batch
insert into Students_info (Roll_No, Studname, Date_of_joing, last_exam_percent)
values(1, 'Asha', '2012-03-12', 79.7)
Values(2, 'Kiran', '2012-03-12', 89.9)
values(3, 'Tarun', '2012-03-12', 78.9)
values(4, 'Samrath', '2012-03-12', 90.9)
values(5, 'Sanjna', '2012-03-12', 67.9)
values(6, 'Rohan', '2012-03-12', 56.9)
Apply batch.

- Select * from Students_info.

DATE: PAGE:

- select * from students_info where Roll_NO IN (1,2,3);
- select * from students_info where StudName='Asha';
- create index on students_info(StudName);
- select * from students_info where StudName='Asha';
- select Roll_NO, StudName from students_info limit 2;
- select Roll_NO as USN from students_info;
- update students_info set StudName='David Shaw' where Roll_NO=2;
- update students_info set Roll_NO=6 where Roll_NO=3;
- delete last_exam_percent from student_info where Roll_NO=2;
- delete from students_info where Roll_NO=2;
- Alter table students_info Add hobbies set<text>;
- Alter table students_info Add language list<text>;
- update students_info
  set hobbies=hobbies+{'chess', 'Table Tennis'}
  where Roll_NO=1;
- update students_info
  set language=language+['Hindi', 'English']
  where Roll_NO=1;
- select * from students_info.

DATE: PAGE:

counter.

→ Create Table library_book(counter_value counter, book_name varchar, studname varchar, PRIMARY KEY (book_name, stud_name));

→ update library_book set counter_value = counter_value + 1 where book_name = 'BDA' AND stud_name = 'jeet';

→ create table userlogin(userid int PRIMARY KEY, password text);

insert into userlogin(userid, password) values(1, 'inpy') using TTL 30;

select TTL(password) from userlogin where userid = 1;

1/A/25

DATE: PAGE:

8/04/25

Lab 4.

1) create a keyspace named library.

create keyspace library
with replication = {'class' : 'simple strategy', 'replication' : 1};

2) create a column family library_info.
we cannot mix counter and non counter columns.
create table library.library_info(
stud_id int Primary key,
stud_name text,
Book_Name text,
Books_Id int,
Date_of_Issue date.

create table library.Book_counter(
stud_id int,
Books_Name text,
counter_value counter,
PRIMARY KEY(stud_id, Books_name))

3) Insert values into the table using Batch

Begin Batch
Insert into library.library_info(stud_id, stud_name,
Book_name, Books_id, Date_of_Issue)
values (112, 'John', 'BDA', 101, '2025-04-06');
Insert into library.library_info(stud_id, stud_name,
Books_Name, Books_id, Date_of_Issue)
values (113, 'Alice', 'DBMS', 102, '2025-05-07');

DATE: PAGE:

Apply Batch

Begin Batch
update library.Book_counter
set counter_value = counter_value + 1
where stud_id = 112 and Book_name = 'BDA'
update library.Book_counter
set counter_value = counter_value + 1
where stud_id = 112 and Book_Name = 'BDA'
Apply Batch

4) Display details and increase counter.

select * from library.library_info
select * from library.Book_counter

update library.Book_counter
set counter_value = counter_value + 1
where stud_id = 112 and Book_name = 'BDA'

5) Query to show student 112 has taken BDA 2 times.

select counter_value from library.Book_counter
where stud_id = 112 and Book_name = 'BDA'

6) Export column to a csv file.

copy library.library_info to 'library_info.csv'
with header = true;

DATE: PAGE:

7) Import csv file to cassandra.

```
copy Library.Library_Info(Stud_Id, Stud_Name,
Book_Name, Book_Id, Date_of_Issue)
from 'Library_info.csv' with header = True.
```

DATE: PAGE:

Lab 5 Hadoop

// start hadoop
$ start-all.sh

// creating a directory inside hadoop - mkdir.
$ hdfs dfs -mkdir /bda_hadoop

// listing all content inside hadoop - ls.
$ hadoop fs -ls /

// copying files from using put command - put.
echo "This is some sample text bda_local.txt file." >
/home/hadoop/bda_local.txt
hdfs dfs -put /home/hadoop/bda_local.txt /bda_hadoop/
file.txt

// cat command (listing the content of file in hadoop) - cat.
hdfs dfs -cat /bda_hadoop/file.txt

// copying files from local reference.
hdfs dfs -copyFromLocal /home/hadoop/bda_local.txt /
bda_hadoop/file_cp_local.txt
hdfs dfs -cat /bda_hadoop/file_cp_local.txt.

// get command.
hdfs dfs -get /bda_hadoop/file.txt /home/hadoop/
downloads/downloaded_file.txt.
hdfs dfs -getmerge /bda_hadoop/file.txt /bda_hadoop/
file_cp_local.txt /home/hadoop/downloads/downloaded_
file.txt.

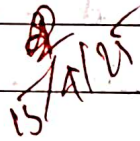
DATE: PAGE:

hadoop fs -getfacl /bda-hadoop/

//copy to local
hdfs dfs -copyToLocal /bda-hadoop/file.txt /home/hadoop/Desktop

//mv
hadoop fs -mv /bda-hadoop/abc.
hadoop fs -ls /abc

hadoop fs -cp /hello/ /hadoop-lab



Lab 6,7

DATE: PAGE:

Mapper

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Mapper;

public class Mapper extends Mapper<LongWritable, Text,
                Text, IntWritable> {
    public static final int MISSING = 9999;

    public void map(LongWritable key, Text value, Mapper
<LongWritable, Text, Text, IntWritable>.Context context)
throws IOException, InterruptedException {
        int temperature;
        String line = value.toString();
        String year = line.substring(15, 19);
        if (line.charAt(87) == '+') {
            temperature = Integer.parseInt(line.substring(88, 92));
        } else {
            temperature = Integer.parseInt(line.substring(87, 92));
        }
        String quality = line.substring(92, 93);
        if (temperature != 9999 && quality.matches("[01459]")) {
            context.write(new Text(year), new IntWritable(
                    temperature));
        }
    }
}
```

DATE: PAGE:

Driver:

```
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class Driver{
    public static void main(String[] args) throws Exception{
        if (args.length != 2){
            System.err.println("Please Enter the input and output
                                parameters");
            System.exit(-1);
        }
        Job job = new Job()
        job.setJarByClass(Driver.this.class);
        job.setJobName("Max temperature");
        FileInputFormat.addInputPath(job, new Path(args[0]));
        FileOutputFormat.setOutputPath(job, new Path(args[1]));
        job.setMapperClass(Average~~s~~Mapper.class);
        job.setReducerClass(CO+AReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        System.exit(job.waitForCompletion(true)? 0:1);
    }
}
```

DATE: PAGE:

Reducer

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.Reducer;

public class Reducer extends Reducer<Text, IntWritable,
    Text, IntWritable> {
    public void reduce(Text key, Iterable<IntWritable>
values, Reducer<Text, IntWritable, Text, IntWritable>.Context
context) throws IOException, InterruptedException {
        int max_temp=0;
        int count=0;
        for(IntWritable value: values) {
            max_temp += value.get();
            count++;
        }
        context.write(key, new IntWritable(max_temp/count));
    }
}
```

DATE: PAGE:

20/8/25 Lab-8.

Aggregate

```
scala> val numbers = sc.parallelize(1 to 20, 4)
numbers: org.apache.spark.rdd.RDD[Int] =
ParallelCollectionRDD[18] at parallelize at <console>: 24

scala> numbers.glom().collect()
res22: Array[Array[Int]] = Array(Array(1,2,3,4,5),
Array(6,7,8,9,10), Array(11,12,13,14,15), Array(16,17,18,19,20))

scala> numbers.aggregate(0)(_+_,_+_)
res23: Int = 210
```

Fold operation.

```
Scala> numbers.fold(0)((x,y) => x+y)

Scala> numbers.aggregate(1)(_+_,_+_)
res37: Int = 215

Scala> val numbers = sc.parallelize(1 to 4,2)
numbers: org.apache.spark.rdd.RDD[Int] =
ParallelCollectionRDD[20] at parallelize at <console>: 24

Scala> numbers.glom.collect
res39: Array[Array[Int]] = Array(Array(1,2), Array(3,4))

Scala> numbers.aggregate(0)(_+_,_*_)

Scala> val allprnc = (x:Int) => x+x
```

DATE: PAGE:

Scala> 3fact(3)
Res0 : Int = 6

Lab 9.

Scala> val sampleRDD = SC.parallelize(List(6,7,5,3,10,25,9,20,35))
sampleRDD: org.apache.spark.rdd.RDD[Int] = ParallelCollection
RDD[26] at parallelize at <console>:24

Sala> sampleRDD.collect
res55: Array[Int] = Array(6,7,5,3,10,25,9,20,35

Lab-10.

→ write a scala program to print numbers from 1 to 100 using for loop.

→ sudo apt install scala.
→ nano PrintNumbers.scala

```
object PrintNumbers {
    def main(args: Array[String]): Unit = {
        for (i <- 1 to 100) {
            println(i)
        }
    }
}
```

→ scalac PrintNumbers.scala
→ Scala PrintNumbers.

→ Using RDD and FlatMap count how many times each word appears in a file and write out a list of words whose count is strictly greater than 4 using spark

DATE:

PAGE:

- mkdir -p wordcountsout/src/main/scala
  cd wordcountsout

- nano build.sbt

```
name := "wordcountsout"
version := "0.1"
scalaVersion := "2.12.17"
libraryDependencies ++= Seq(
  "org.apache.spark" %% "spark-core" % "3.3.1"
)
mainClass in compile := Some("wordcountsout")
```

- nano src/main/scala/wordcountsout.scala

- echo "hello spark hello spark hello world spark
  spark" > /home/bhoom/Desktop/wc.txt

- sbt run

```
import org.apache.hadoop.io.IntWritable
import org.apache.hadoop.io.Text
import org.apache.hadoop.mapreduce.Reducer
import java.util.*;
public class WordcountReducer {
    private final IntWritable result = new IntWritable()
    public void reduce() throws IO exception {
        int sum=0;
        for (IntWritable val:values) {
            sum += val.get();
        }
```

DATE: PAGE

result.setSum)
context.write(key, result);
}

20/5/25