INDEX

Name: Nidhi. A Std: ____ Sec: ____

Roll No. ____ Subject: BDA School/College: ____

School/College Tel. No. ____ Parents Tel. No. ____

Sl. No.	Date	Title	Page No.	Teacher Sign / Remarks
1.	4/3/25	Lab 1		
		lab		
2.	11/3/25	Lab 2		
3.	04/04/25	Lab - 3		
4.		Lab - 4		
5.		Lab - 5		
6.	15/4/25	Lab - 6		15/4/25
7	22/4/25	Lab - 7		
8	28/4/25	Lab - 8		
9.	2/5/25	Lab - 9		

Date / /
Page

Lab

① use myDB;
② db;
③ show dbs;

admin	40.00 KB
config	40.00 KB
local	84.00 KiB
myDB	72.00 KiB

④ db.createCollection("Student");
{ ok : 1 } dropped
⑤ db.Student.drop();
{ ok : 1 } dropped
6. db.createCollection("Student");
{ ok : 1 }
⑦ db.Student.insert({ _id: 1, studname: "Nidhi", grade: "VII", Hobbies: "surfing" });

{ acknowledged: true, insertedIds: {'0': 1}}

⑧ db.Student.update({ _id: 3, studName: "AryanDawid", Garade: "VII"}, { $set: { Hobbies: "Skating" }}, { upsert: True});

{ acknowledged: True,
insertedId: 3
matchedCount: 0,
modifiedcount: 0,
upsertedcount: 1 }

Date __/__/__
Page _____

(9) db.Student.find({ StudName: "AryanDavid"})

{ _id: 3, Grade: 'VII', StudName: 'AryanDavid',
Hobbies: 'Skating' }

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

[{ }, { Grade: 'VII', StudName: 'AryanDavid' }]

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

{ { id: 3, Grade: 'VII', StudName: 'AryanDavid'

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

{_id: 3, Grade 'VII', StudName: 'AryanDavid'
Hobbies: 'Skating' }

(13) db.Student.find({studname: /^A/ }).pretty();

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

(15) db.Student.count();

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

VI (17) db.Student.update({_id:4}, {$set: { Location:
"Networks" }});

{ ack: true,
insertedID: 4,
matchedCount: 0,
modified count: 0,
upsertedCount: 0}

Date __/__/__
Page _____

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

IX db.Student.find({Grade:{$ne:'VII'}}).pretty()
[{_id: 4}]

X db.Student.update({_id:3}, {$set:{Location: null}});

{acknowledged: true,
insertedId: null,}

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

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

XIII db.food.insert({_id:1, fruits:['grapes','mango','apple']})
{ok: 1}
{acknowledged: true, insertedIds: {'0': 1}}

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

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

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

Date ___/___/___
Page _____

Aggregate function

```
1  db.customers.aggregate([{
        $group: {
            _id: "$cust ID",
            TotaccBal: {$sum: "$AccBal"}
        }
    }]);
```

```
db.customers.aggregate([{$match: {AccType: "S"}
    },
    {$group: {_id: "$custID",
            TotAccBal: {$sum: $AccBal"}
        }
    }]),
```

```
db.customers.agregate({$match: {AccType: "S"}
    , {$group: {_id: $custID, TotAccBal,
    {$sum: "$AccBal"}}},
        {$match: {TotAccBal: {$gt: 1200}}}])
```

14/3/2024

OUTPUT

```
[{"_id": "C001", "TotalAccBal": 1500},
  {"_id": "C004", "TotAccBal": 1500}
]
```

```
[{_id: "C004", "TotaccBal: 1500},
  {_id: "C001", TotalAccBal: 1200},
  {_id C003", TotaccBal: 500}
]
```

11/03/25

Date __/__/__
Page _____

Lab - 02

I. MongoDb Lab Exercise: Perform the following operations using the MongoDB

Ans use myDb;

1.

1. Create a collection by name customers with the following attributes. Cust-id, Acc-Bal, Acc-Type.

Ans db.createCollection("Customers");

2. Insert at least 5 values into the table

Ans db.Customers.insertMany([{"Cust-id": 1, "Acc-Bal": 1500, "AccType": "z"},
{"Cust-id": 2, "Acc-Bal": 1800, "Acc-Type": "z"},
{"Cust-id": 3, "Acc-Bal": 500, "Acc-Type": "x"}]);

Output:
{ acknowledged: true,
InsertedIds: { '0': ObjectId('67...14'), ...} }

3. Write a query to display those records whose total account balance is greater than 1200 of ~~curre~~ acc type "z".

Ans. db.Customers.find({"Acc-Bal": {$gt:1200},
"Acc-Type": "z"});

[{ _id: ObjectId('67cff...'),
Cust-id: 1,
Acc-Bal: 1500,
Acc-Type: 'z' },

Date __/__/__
Page

```
{ _id: ObjectId('67 ... 15'),
  Cust_id: 4
  Acc-Bal : 1300
  Acc-Type : 'Z' } ]
```

4. Determine Minimum and Maximum account balance for each customer_i:

```
db.Customers.aggregate([
    { $group: {
        _id: "$cust-id",
        min_bal: {$min: "$Acc-Bal"},
        max_bal: {$max: "$Acc-Bal"}
    }} ]);
[
  {_id: 2, min-bal: 1800, max-bal: 1800},
  {_id: 4, min-bal: 1300, max-bal: 1300},
  {_id: 3, min-bal: 500, max-bal: 500}
]
```

//OUTPUT//

(2) You are developing ..... and orders.

Query to:

1. Retrieve all Products.

Ans. db.Product.find({});

Output

```
[ { _id: ObjectId('67 ... 19'),
    product-id: 'prod001',
    name: 'laptop'
    category: 'Electronics'
    price: 1200
    quantity: 10 },
```

Date ___/___/___
Page _____

{ -id ..... }
]

2. Retrieve Products in a category specific (eg: Electronics)

Output:

db.Product.find({ "category" : "Electronics" });

[{
id : ObjectId('67cff ... 19')
product-id : 'prod001',
name : 'Laptop',
category : 'Electronics',
price : 1200
quantity : 10 },]

3. Retrieve products with quantity greater than 0

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

Output

[{ id : ObjectId('67 ... 19')
product-id : 'prod001',
name : 'Laptop'
category : 'Electronics',
price : 1200
quantity : 10 },
{ id : ObjectId('67 ... 20')
product id : 'prod002'
name : 'Phone'
category : 'Electronics'
price : 5000
quantity : 20 }
]

Date __/__/__
Page

(4) Sort products Asc on price

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

(5) With Price <=100

```
db.Products.find({
    "price": {$lt:100}
});
```

(6) Product Added to users cart

```
db.Carts.findOne({"user-id":"789ghi"},
    {"products":1});
```

Output:

```
{_id: ObjectId('67...20')
 products: [
    {product_id: 'prod001', quantity:2}
    {product_id: 'prod002', quantity:3}
 ]
}
```

(7) Order placed by user

```
db.Orders.find({
    "user_id": "123abc"
});
```

Output:

```
[ {_id: ObjectId('...22')
   userid: '123abc'
   total price: 1600
}
]
```

Date __/__/__
Page _____

(8) Total price by the user Orders

```
db.Orders.aggregate([
    { $match: { "user_id": "123abc" } },
    { $group: { _id: "user_id", tot_price{ $sum:
        "$ tot_price" } } }
    ]);

[ _id: '123abc, total_price: 1600 }]
```

Additional Aggregation

1. Calculate total number of Product in each category

```
db.ag Products.aggregate([ {$group: { _id:
        "$category", tot_product: { $sum: 1 } }
    ]);
[
  { _id: 'Furniture', tot_products: 2 },
  { _id: 'Clothing', tot_products: 2 },
  { _id: 'Electronics', tot_products: 3 } ]
```

Output

2. Calculate total ~~number~~ Price of Product in Each Category.

```
db.Product.aggregate([
    { $group: {
        _id: "$category",
        tot_price: { $sum: { $multiply:
            ["$price", "$quantity"] }
    } } } ]);
```

Date __/__/__
Page _____

[{ id: Electronics , total-price: 2000 },
{ -id: 'Furniture', total-price: 1300 },
{ -id: 'Clothing', total-price: 2450 }
]

③ Find Average Price of Products.

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

[{ _id, null, avg_price: 359.285 }]

④ Find Products with Quantity less than 10

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

[_id ...
product_id ...
name: "Chair"
category: "Furniture"
price: 100
quantity: 5
}]

⑤ Calculate Total Price of Orders.

```
db.Orders.aggregate([ { $group: {
    _id: "$user_id
```

Date __/__/__
Page

```
total_spent: { $sum: "$ tot_price" }
}
} } ]);
```

```
[ {id: '123abc', total_spent: 1600},
  {id: '789ghi', total_spent: 2500} ]
```

7. Find user with highest total price of order.

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

```
[ id: '789ghi', total_spent: 2500} ]
```

8. Find Average Total Price of Orders.

```
db.Orders.aggregate([
  { $group: { _id: null, avg_tot_pri:
      $avg: "$ tot_price" }
  } } ]);
```

```
[ { _id: null, avg_price: 2050} ]
```

11/3/25

01/04/25

Date __/__/__
Page

Lab - 03

① Create Keyspace students with replication = {'class' = SimpleStrategy, 'replication_factor' : 1}

② Describe keyspaces

students system_auth system_schema

③ SELECT * FROM system_schema.keyspaces;

④ USE students;

⑤ CREATE TABLE Students_Info (Roll_no int PRIMARY KEY, Studname text, DateOfJoining timestamp, last_exam_percent double);

⑥ DESCRIBE TABLE Students_Info;

⑦ Insert values
BEGIN BATCH
Inset into Students_info (Roll_no, StudName, DateOfJoining, last_exam_Percent) values (1, 'Asha', '2012-03-12', 79.9)
Insert into Students_info (Roll_no, StudentNm, DateOfJoing, last_exam_Percent) values (2, 'Jasem', '2012-03-12', 90.9)
⋮
APPLY BATCH;

8. Select * from Students_Info;

Date ___/___/___
Page _____

roll_no	date of joining	last exam percent	studname
5	2012-03-21 18:30:00	67.9	Smitha
1	"	79.9	Asha
2	"	89.9	Kiran
3	"	90.9	Tarun

(9) SELECT * FROM STUDENTS_info WHERE Roll_no IN (1, 2, 3);

roll_no	date of joining	last exam percent	Studname
1	2012-03-11	79.9	Asha
2	"	89.9	Kiran
3	"	78.9	Tarun

(10) CREATE INDEX ON Students_info (StudName)

(11) Select * from Student_Info where Studname = "Asha"

rollno	date of joinig	last exam percent	Studnam
1	2012-03-11	79.9	Asha

(12) Select Roll_no as "USN" from Student_info

USN
5
1
2
3

(13) Select USN, Student from Delete_limit_2;

rollno	Student
1	Smitha
2	Asha

(14)

Date __/__/__
Page

UPDATE

UPDATE student_info SET studname = 'David Sheen' WHERE roll_no = 2

~~UPDATE~~
DELETE LastExamPerf FROM student_info WHERE Roll_no = 2

ALTER TABLE students_info ADD Hobbies set <text>

UPDATE student_info SET hobbies = hobbies + {'Chess', 'Table Tennis'} where Roll no = 1;

CREATE TABLE lib_book (counter_val, counter, book name ''varchar, PK (bookname, stud_name))

Loading data

Update lib_book SET counter_value = counter_value WHERE book name = 'BDA'.

Time to Live

Create table login (userid int PRIMARY KEY, p text)

INSERT INTO login (userid, p) values (1, 'abc') USING TTL = 30;

SELECT TTL (password) FROM login where user_id = 1

Export to csv

copy elearnslist (id, courseorder, course_id)
TO 'd:\elearning.csv');

Import to csv.

COPY elearning (id, course_order, course_name,
title) FROM 'd:\elarnslist.csv'

11/4/2025

Date __/__/__
Page

Lab - 4

① CREATE a keyspace by name Library

```
CREATE KEYSPACE Library WITH REPLICAT
= {'class': 'SimpleStrategy', 'replication factor
};
```

② create column family library_info

```
CREATE TABLE library_counter_Table(
    stud_id in PRIMARY KEY,
    counter_value counter
);
```

3

```
CREATE TABLE library.Library_info(
    stud_id int,
    stud_name text,
    Book_name text
    Book_id int,
    Date of issue 'date);
```

③ Insert value into table in batch

```
BEGIN BATCH
    INSERT INTO library.Library_info(
Stud_id, stud_name, book_name, book_id,
Date of issue)
values (113, 'BOb', 'PBMS', 'B102', '2025-04-05
);
APPLY BATCH;
```

```
Update library.counter_table SET counter
= counter_val + where stud_id = 112;
```

Date ___/___/___
Page _____

6 Display & update counter

Stud id	book name	book id	date of issue	stud name
113	DBMS	B102	15-04-02	Bob
112	BDA	B101	15-04-01	Alice

7:

stud id	counter value
112	1

5s show a student with ID 112 has taken book "BDA" 2 times

SELECT * FROM library counter table where stud-id = 112;

Stud ID	counter value
112	2

3b Export table to CSV

exit cqlsh;
cqlsh: "copy library library info to 'library info .csv' WITH HEADER = 'true'.
import data from csv
cqlsh -e "

(7) Show import data
USE library;
SELECT * from library info.

sid	b-N	b-id	d-of issue	stu name
113	DBMS	B102	25-04-02	Bob
112	BDA	B101	25-04-01	Alice

15/04/25

Date __/__/__
Page

Lab - 06

① Start command

start-dfs.sh

② hdfs dfs -mkdir /alec

③ hadoop fs -ls /

Found 12 items

drwxr-xr-x hadoop supergroup 0 2023/08 [illegible] /[illegible]
" " " " /alec
" " " " /user
" " " " /wc-[illegible]

④ mkdir -p /home/hadoop/Desktop
echo "Hello Hadoop World" > /home/hadoop/[illegible]
/Welcome.txt

⑤ hdfs dfs -put /home/hadoop/Desktop/Welcome
/alec/wc.txt

⑥ hdfs dfs -copyfromlocal -f /home/hadoop/Desktop
/Welcome.txt /alec/wc.txt

⑦ mkdir -p /home/hadoop/Desktop

⑧ hdfs dfs -getmerge /alec/wc.txt
/home/hadoop/Desktop/Merge.txt

Date / /
Page

hadoop fs -getfacl /abc/
file : /abc
owner : hadoop
group : supergroup
user : rwx
group : r-x
other : r-x

(9) hdfs dfs -cat /abc/wc.txt
Hello Hadoop World!

(10) hadoop fs -mv /abc/FFF

(11) hadoop fs -cp /pooja34/ /FFF/
(12) ls/4

Date __/__/__
Page

Lab 07 - Word count.

WCDriver.java.

```
import  java.io.IOException;

public  class  WCDriver extends Configured implements
 Tool {
      public  int  run (String  args[]) throws
       IOException {
            if (args.length < 2) {
                 System.out.println(" Please give
                  valid inputs ");
                return -1;
            }

            JobConf  conf = new JobConf (WCDriver.class);
            FileInputFormat.setInputPaths (conf, new
                 Path(args[0]));
            FileOutputFormat.setOutput(conf, new Path
                 (args[0]));
            conf.setMapperClass(WCMapper.class);
            conf.setReducerClass(WCReducer.class);
            conf.setMapOutputKeyClass (Text.class);
            conf.setOutputValueClass (IntWritable
                                        class).
            conf.setOutput (IntWritable.class).
            JobClient.run(conf);
            return 0;
      }
      public static void main (String args[])
       throws Exception {
            int exitCode = ToolRunner.run(
```

Date / /
Page

MAPPER.JAVA

```
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.hadoop.mapred.MapReduceBase;
import org.apache.hadoop.mapred.OutputCollector;

public class WCMapper extends MapReduceBase
implements Mapper < LongWritable Text, Text, IntWritable>
{ output, Reporter rep) throws IOException {
        String line = value.toString();
        for (String word : line.split(" ")) {
            if (word.length() > 0){
                output.collect(new Text(word),
                new IntWritable(1));
            } }
    }
```

REDUCER.JAVA

```
import java.io.IOException;
import java.util.Iterator;
import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.mapred.Reporter;

public class WCReducer extends MapReduceBase
implements Reducer< Text, IntWritable, Text, IntWritable>
{ public void reduce (Text key, Iterator<IntWritable>
        value, OutputCollector < Text, IntWritable > output
        Reporter rep) throws IOException {
        int count = 0;
        while (value.hasNext()){
```

```
            IntWritable i = value.next();
            count += i.get();
        }
        output.collect(key, new IntWritable(count));
    }
}
```

Date __/__/__
Page _____

Lab - 08 - MeanMax

```
package meanmax;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IntWritable;

public class MeanMaxDriver {
    public static void main (String[] args)
    throws Exception {
        if (args.length != 2) {
            System.err.print("In OP parameter");
            System.exit(-1);
        }
        Job job = new Job();
        job.setJarByClass(MeanMaxDriver.class);
        job.setJobName("Max temperature");
        FileInputFormat.setOutputPath(job, newPath(args[0]));
        job.setMapperClass(MeanMaxMapper.class);
        job.setOutputKeyClass(Text.class)
        job.setOutputValueClass(IntWritable.class);
        System.exit(job.waitForCompletion(true)?0:1);
    }
}

package meanmax;
import java.io.IOException;
import org.apache.hadoop.io.IntWritable;

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

```
public void map(LongWritable key, Text
Mapper<LongWritable, Text, Text, IntWritable>.
Context, context) throws IOException, Inter
Exception{
    int temp;
    String line = value.toString();
    String month = line.substring(19, 21);
    if(line.charAt(87) == '+'){
        temp = Integer.parseInt(line.substring(87
    }else {
        temp = Integer.parseInt(line.substring(87,8
    }
    String quality = line.substring(92, 98);
    if (temperature != 9999 && quality.matches("[0
        context.write(new Text(month), new
            IntWritable(temp));
    }
}
```

```
package meanmax;

import java.io.IOException;
import .....
;
import org.apache

public class MeanMaxReducer extends Reducer<Text,
IntWritable, Text, IntWritable> {
    public void reduce(Text key, Iterable<IntWritable
        values, Reducer<Text, IntWritable, Text, IntWrit
        , Context context) throws IOException, {
        int maxtemp = 0;
        int total temp = 0;
```

Date ___/___/___
Page _____

```
int count = 0;
int days = 0;
for (IntWritable value: values) {
        int temp = value.get();
        if (temp > max_temp)
            max_temp = temp;
        count++;
        if (count == 3) {
            total_temp += max_temp;
            maxtemp = 0;
            count = 0;
            days++;
        }
    }
    context.write(key, new IntWritable(total_temp/
        days));
}
}
```

Date __/__/__
Page

Lab - 9

6. Write a scala program to print from 1 to 100 using for loop

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

9

Using RDD & Flatmap, count how many time each word appears in a file & write out a list of words whose count is strictly greater than 4 using spark

```
import org.apache.spark.{SparkConf, SparkContext}

object WordCount {
        def main (args: Array[String]): Unit = {
                val conf = new SparkConf () setAppName
                        ("WordCount") setMaster("local[*]")
                val sc = new SparkContext (conf)

                val input = sc.textFile ("desktop/ac.text")
                val word = input
                        .flatmap(line => line.split(" \w+"))
                        .filter (_ .nonEmpty)
                        .map (_ .tolowerCase)
                        .map(word => (word, 1))
```

```
.reduceByKey (_ + _)

    val filteredwords = words.filter(cast, correct)
    filteredwords.collect.foreach {
        case(word, count) => println(s"$
        word : $count")
    }
    sc.stop()
  }
}
```

Q. For a given text file, create a map reduce program to sort the content in an alphabetical order listing only top 10 maximum occurances of words.

⇒ ~~mapper.py~~

```
import sys
import re
for line in sys.stdin:
    words = re.findall(r'[a-zA-Z]+', line.lower())
    for word in words:
        print(f"{word}\t1")
```

reducer.py

```
import sys
from collections import defaultdict
word_count = defaultdict(int)
for line in sys.stdin:
    word, count = line.strip().strip('\t')
    word_count[word] += int(count)
```

Date ___/___/___
Page _____

```
sorted_words = sorted (word_count.items(), key= lambda
                    n : (n[1], n[0]))
for words, count in sorted_words [:10]:
    print(f"{word}\t{count}");
```

Create a text.txt file & enter some content

→ hadoop jar /home/hadoop/.....
lib/hadoop-streaming-3.3.4.jar.
- input file : ///home/hadoop/test.txt
- output :
- mapper :
- reducer :

20/3/2025