**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

**“JnanaSangama”, Belgaum -590014, Karnataka.**

****

**LAB REPORT**

**on**

**Big Data Analytics (23CS6PCBDA)**

***Submitted by***

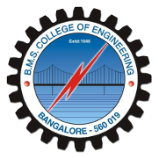
**Krishna Mehta(1BM22CS130)**

***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**

**Feb-2025 to June-2025**

**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 (23CS6PCBDA)” carried out by Krishna Mehta (1BM22CS130), who is bonafide student of B. M. S. College of Engineering. It is in partial fulfilment for the award of Bachelor of Engineering in Computer Science and Engineering of the Visvesvaraya Technological University, Belgaum during the year 2025. The Lab report has been approved as it satisfies the academic requirements in respect of a Big Data Analytics - (23CS6PCBDA) work prescribed for the said degree.

Sneha P                Dr. Kavitha Sooda

Assistant Professor Professor and Head

Department of CSE Department of CSE

BMSCE, Bengaluru BMSCE, Bengaluru

**Index Sheet**

|  |  |  |
| --- | --- | --- |
| **Sl. No.** | **Experiment Title** | **Page No.** |
| 1 | MongoDB- CRUD Demonstration. | 1-4 |
| 2 | Perform the following DB operations using Cassandra.  a) Create a keyspace by name Employee  b) Create a column family by name Employee-Info with attributes Emp\_Id Primary Key, Emp\_Name, Designation, Date\_of\_Joining, Salary,Dept\_Name  c) Insert the values into the table in batch  d) Update Employee name and Department of Emp-Id 121  e) Sort the details of Employee records based on salary  f) Alter the schema of the table Employee\_Info to add a column Projects which stores a set of Projects done by the corresponding Employee.  g) Update the altered table to add project names.  h) Create a TTL of 15 seconds to display the values of Employees. | 5-6 |
| 3 | Perform the following DB operations using Cassandra.  a) Create a keyspace by name Library  b) 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  c) Insert the values into the table in batch  d) Display the details of the table created and increase the value of the counter  e) Write a query to show that a student with id 112 has taken a book “BDA” 2 times.  f) Export the created column to a csv file  g) Import a given csv dataset from local file system into Cassandra column family | 7-8 |
| 4 | Execution of HDFS Commands for interaction with Hadoop Environment. (Minimum 10 commands to be executed) | 9-10 |
| 5 | Implement Wordcount program on Hadoop framework | 11-14 |
| 6 | From the following link extract the weather data https://github.com/tomwhite/hadoop book/tree/master/input/ncdc/all Create a Map Reduce program to a) find average temperature for each year from NCDC data set. b) find the mean max temperature for every month. | 15-23 |
| 7 | For a given Text file, Create a Map Reduce program to sort the content in an alphabetic order listing only top 10 maximum occurrences of words. | 24-30 |
| 8 | Write a Scala program to print numbers from 1 to 100 using for loop. | 31 |
| 9 | 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. | 32-33 |
| 10 | Write a simple streaming program in Spark to receive text data streams on a particular port, perform basic text cleaning (like white space removal, stop words removal, lemmatization, etc.), and print the cleaned text on the screen. (Open Ended Question). | 34-35 |

**Course Outcome**

|  |  |
| --- | --- |
| **CO1** | Apply the concept of NoSQL, Hadoop or Spark for a given task |
| **CO2** | Analyze big data analytics mechanisms that can be applied to obtain solution for a given problem. |
| **CO3** | Design and implement solutions using data analytics mechanisms for a given problem. |

**Experiment-1**

Q) MongoDB- CRUD Operations Demonstration (Practice and Self Study)

**Code & Output:**

1. Create a database “Student” with the following attributes Rollno, Name , Age, ContactNo, Email-Id, grade, hobby:

use Students;

1. Insert 5 appropriate values according to the below queries.

db.students.insertMany([

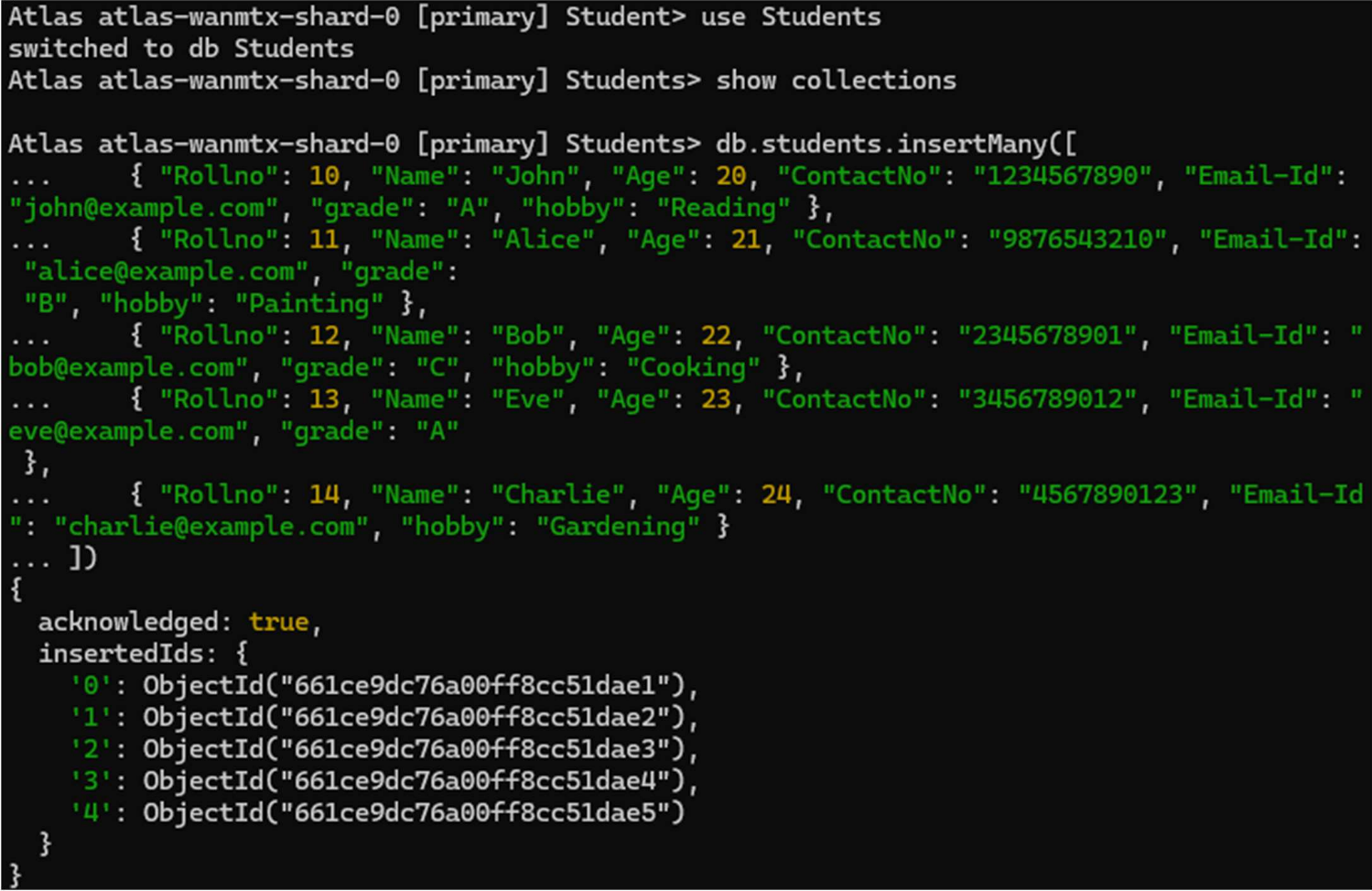
{ "Rollno": 10, "Name": "John", "Age": 20, "ContactNo": "1234567890", "Email-Id": "john@example.com", "grade": "A", "hobby": "Reading" },

{ "Rollno": 11, "Name": "Alice", "Age": 21, "ContactNo": "9876543210", "Email-Id": "alice@example.com", "grade": "B", "hobby": "Painting" },

{ "Rollno": 12, "Name": "Bob", "Age": 22, "ContactNo": "2345678901", "Email-Id": "bob@example.com", "grade": "C", "hobby": "Cooking" },

{ "Rollno": 13, "Name": "Eve", "Age": 23, "ContactNo": "3456789012", "Email-Id": "eve@example.com", "grade": "A" },

{ "Rollno": 14, "Name": "Charlie", "Age": 24, "ContactNo": "4567890123", "Email-Id": "charlie@example.com", "hobby": "Gardening" }])



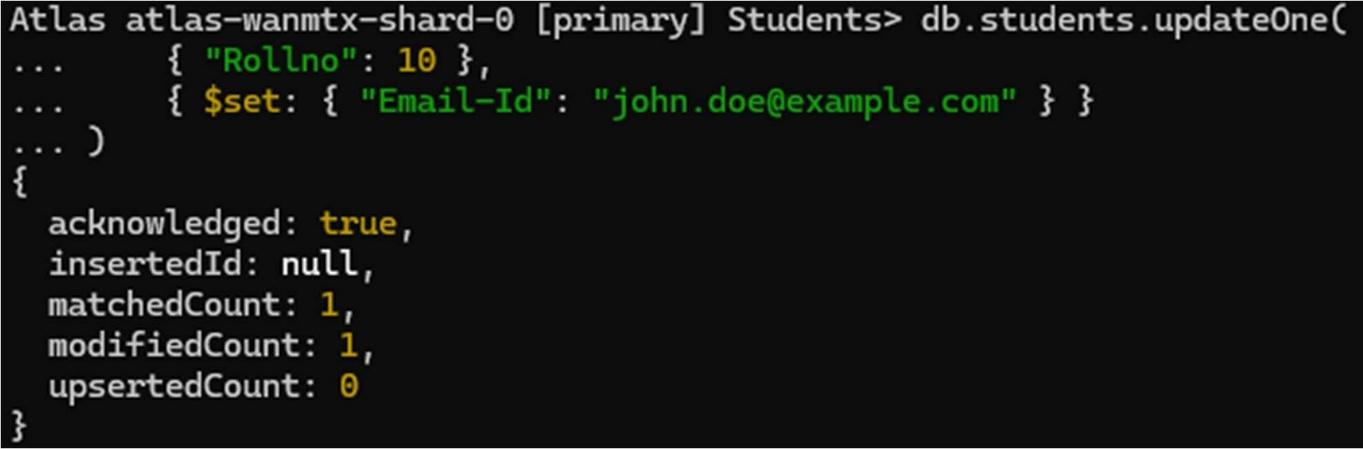
1. Write query to update Email-Id of a student with rollno 10.

db.students.updateOne(

{ "Rollno": 10 },

{ $set: { "Email-Id": "john.doe@example.com" } }

)



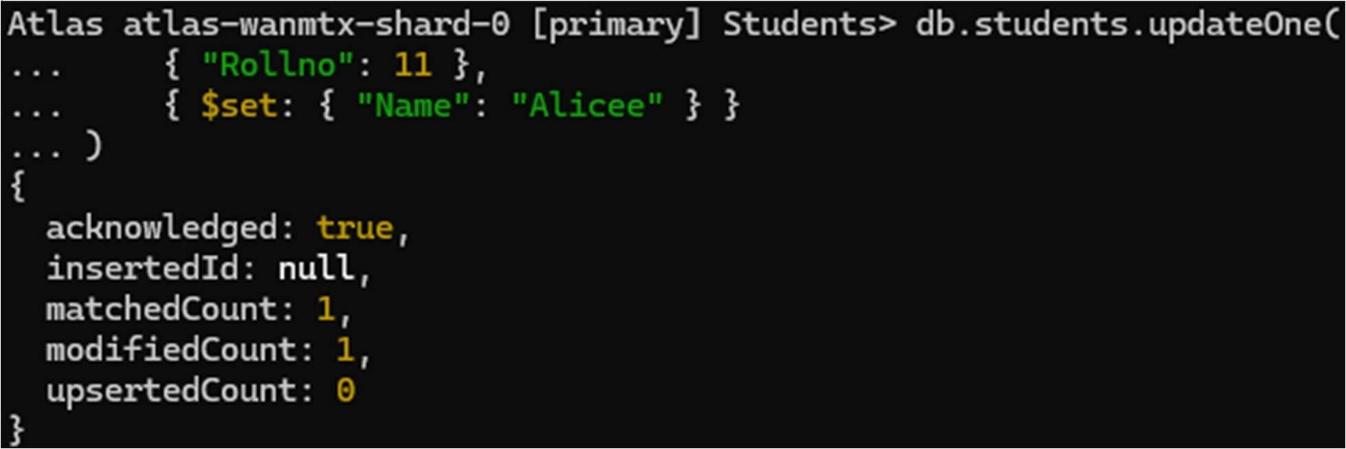
1. Replace the student name from “Alice” to “Alicee” of rollno 11

db.students.updateOne(

{ "Rollno": 11 },

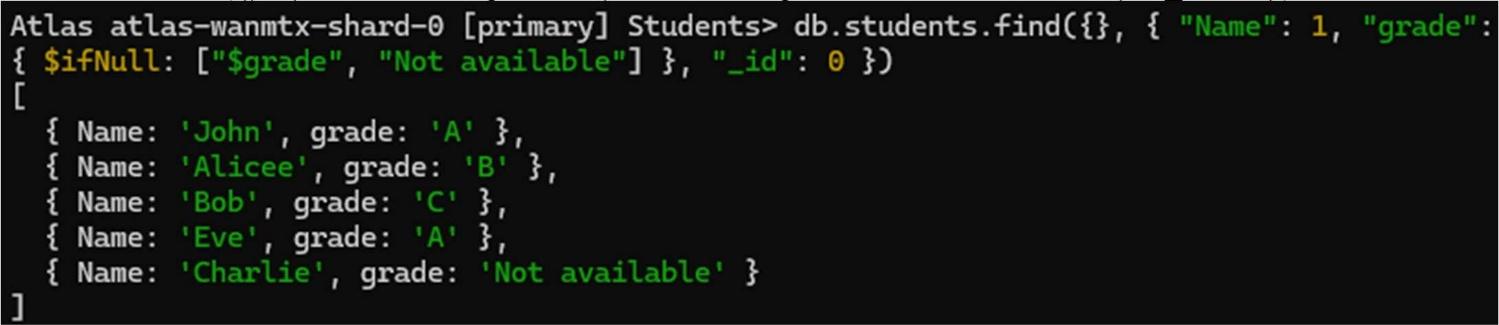
{ $set: { "Name": "Alicee" } }

)



1. Display Student Name and grade(Add if grade is not present)where the \_id column is 1.

db.students.find({}, { "Name": 1, "grade": { $ifNull: ["$grade", "Not available"] }, "\_id": 0 })

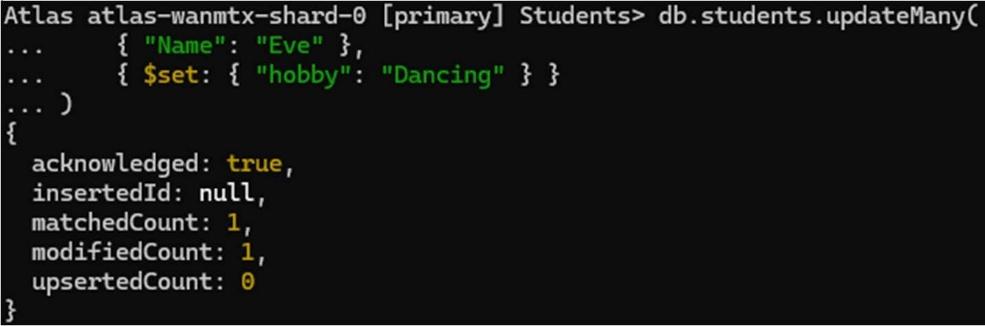


1. Update to add hobbies

db.students.updateMany(

{ "Name": "Eve" },

{ $set: { "hobby": "Dancing" } })



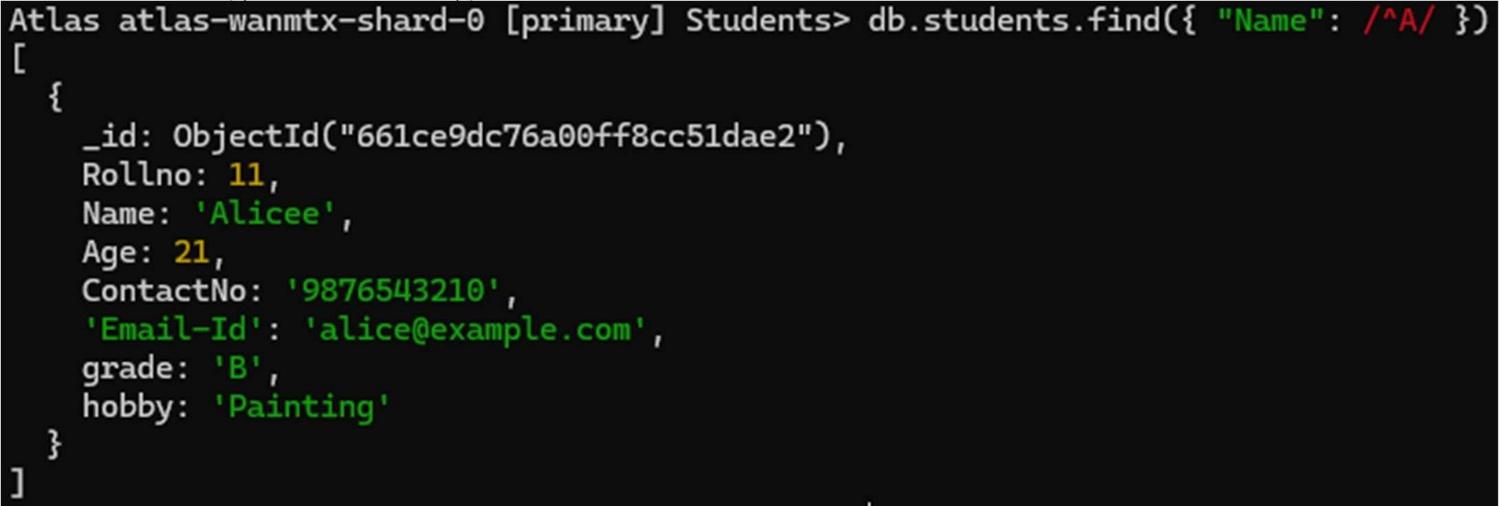
1. Find documents where hobbies is set neither to Chess nor to Skating

db.students.find({ "hobby": { $nin: ["Chess", "Skating"] } })



1. Find documents whose name begins with A

db.students.find({ "Name": /^A/ })



**Experiment – 2**

Q) Perform the following DB operations using Cassandra

a) Create a keyspace by name Employee

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

c) Insert the values into the table in batch

d) Update Employee name and Department of Emp-Id 121

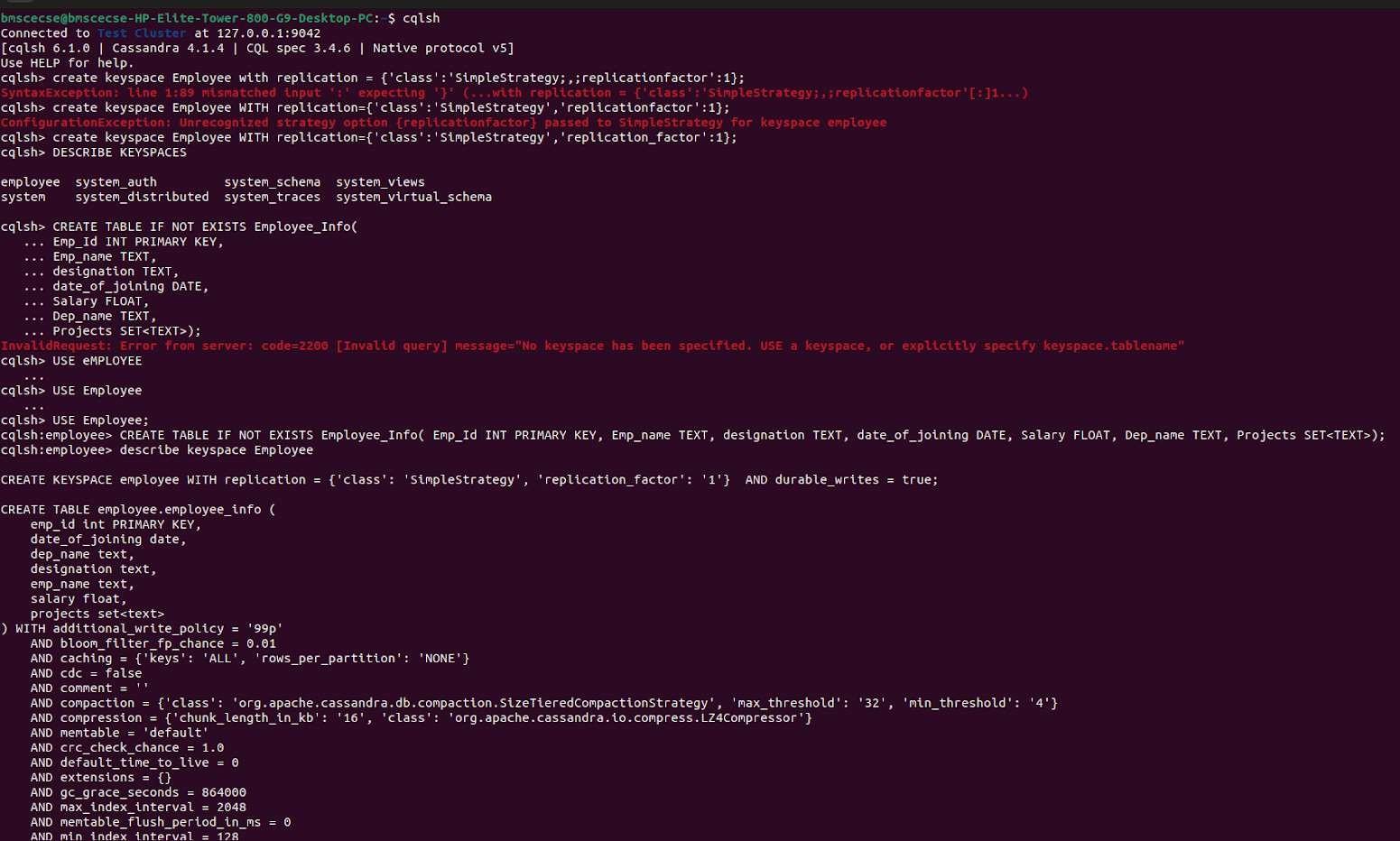
e) Sort the details of Employee records based on salary

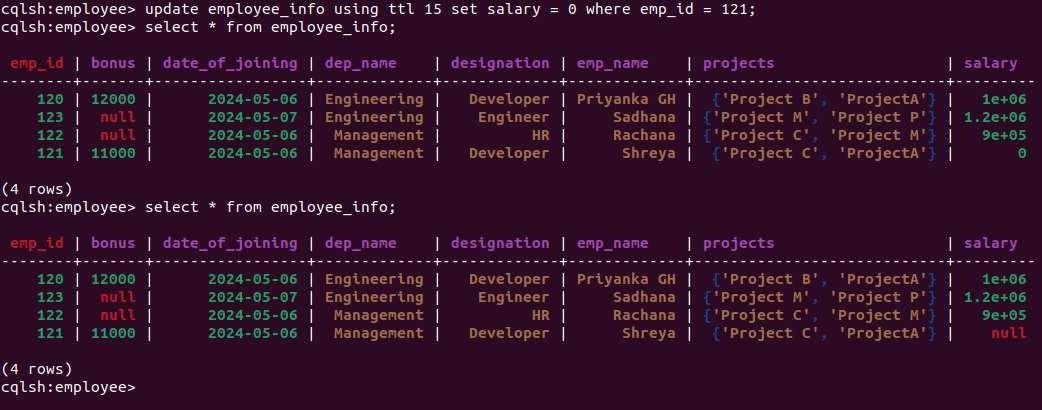
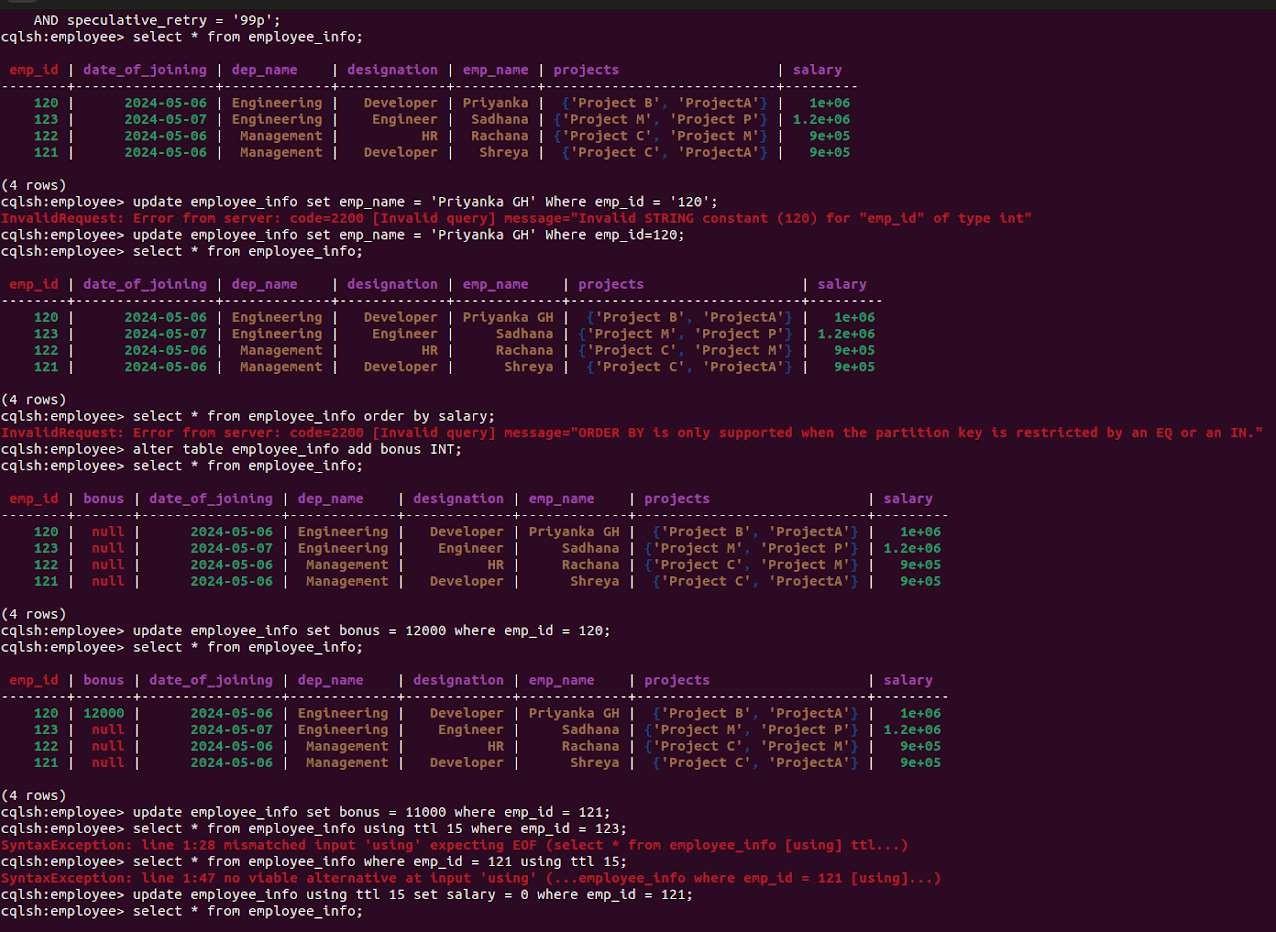
f) Alter the schema of the table Employee\_Info to add a column Projects  
  which stores a set of Projects done by the corresponding Employee.

g) Update the altered table to add project names

h) Create a TTL of 15 seconds to display the values of Employees

**Code & Output:**



\

**Experiment – 3**

Q) Perform the following DB operations using Cassandra

a) Create a keyspace by name Library

b) 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

c) Insert the values into the table in batch

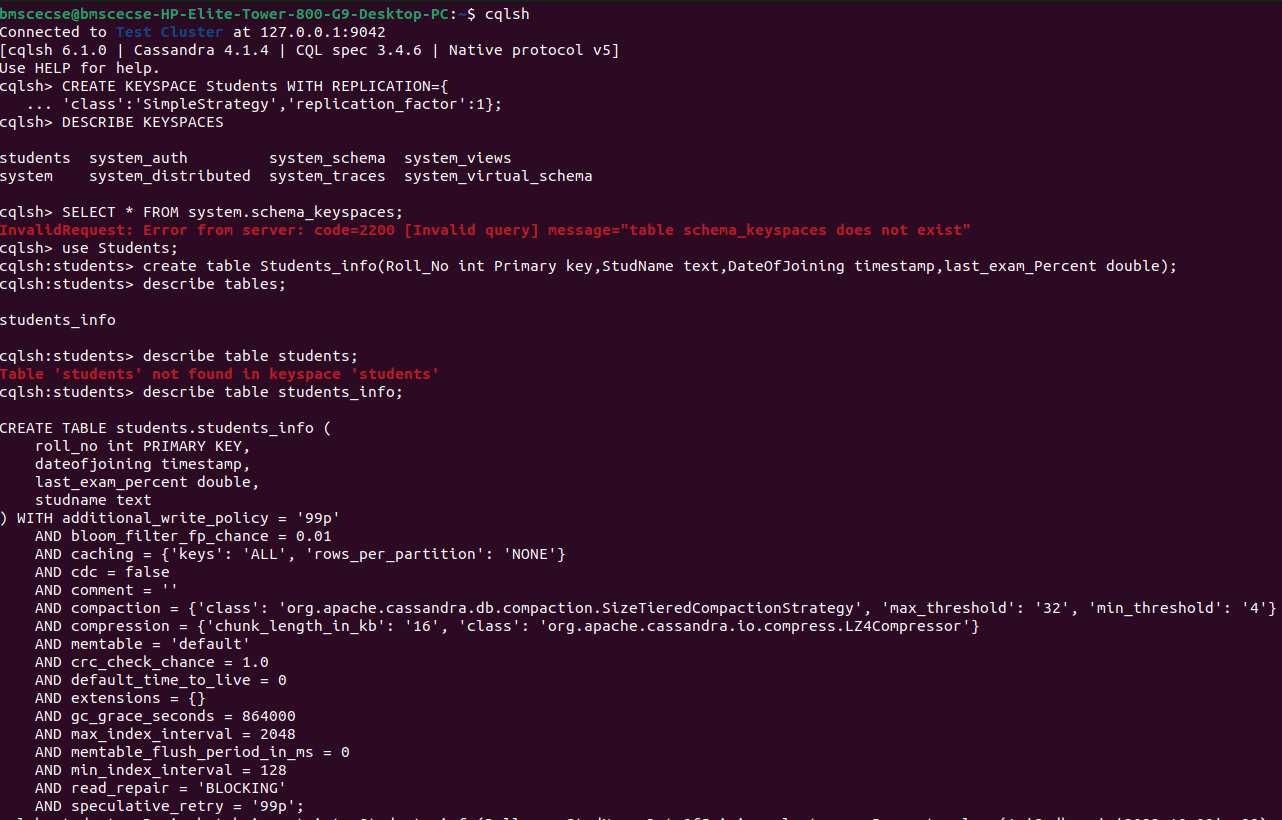
d) Display the details of the table created and increase the value of the counter

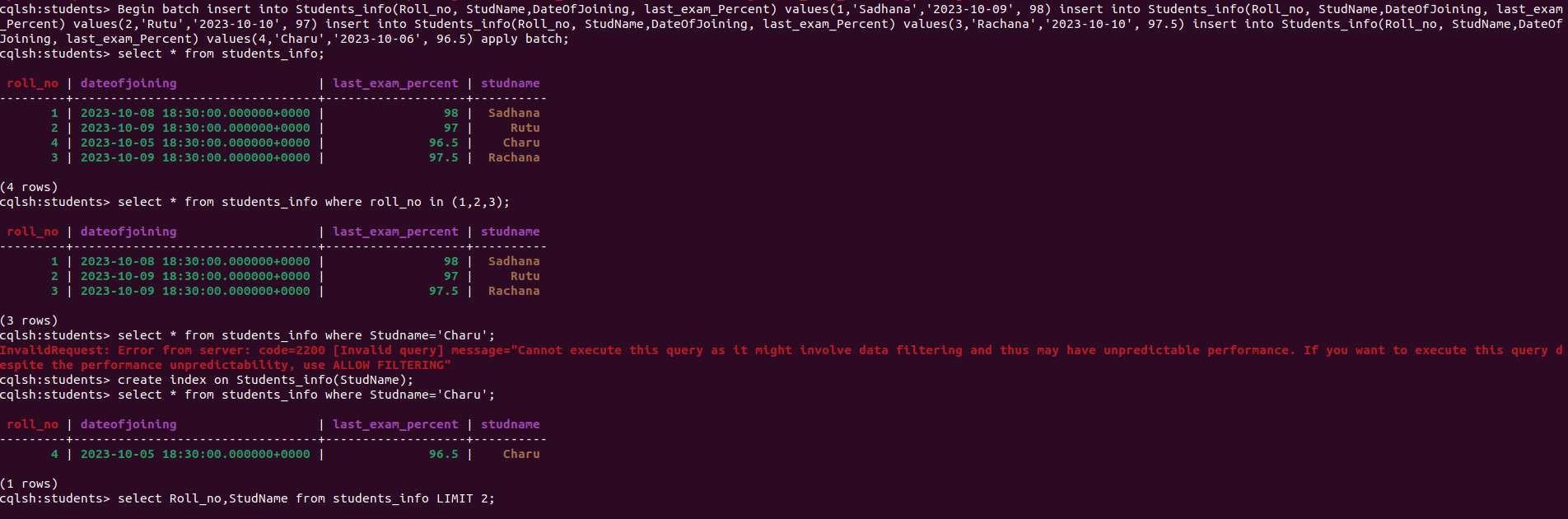
e) Write a query to show that a student with id 112 has taken a book “BDA” 2 times

f) Export the created column to a CSV file

g) Import a given CSV dataset from local file system into Cassandra column family

**Code & Output:**



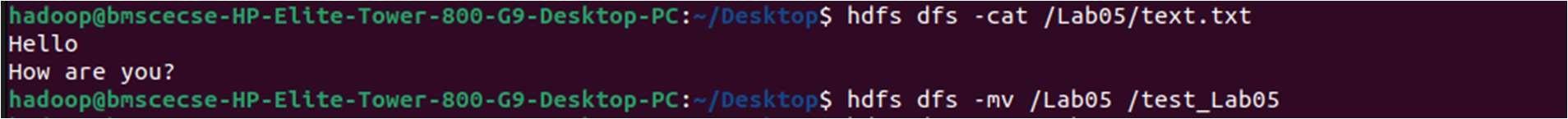
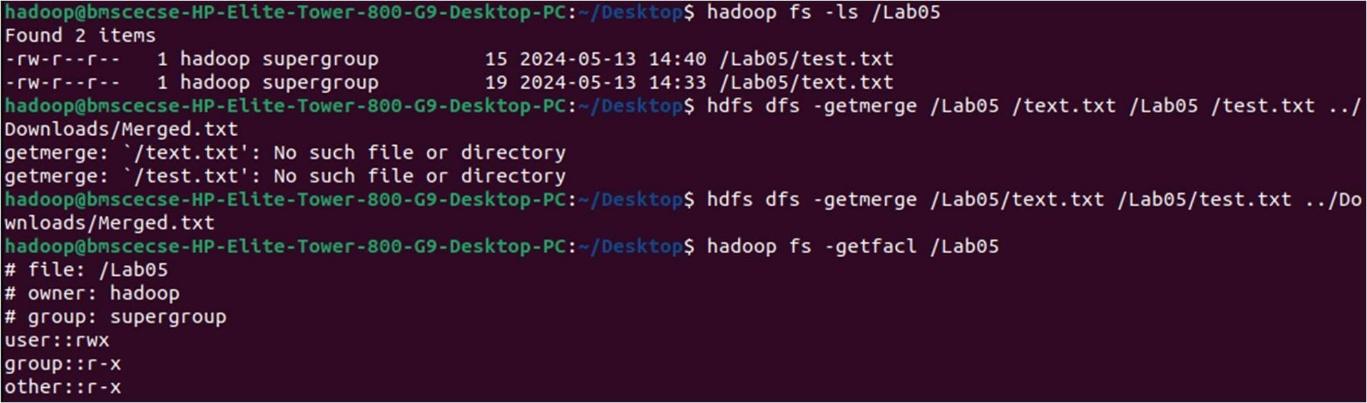
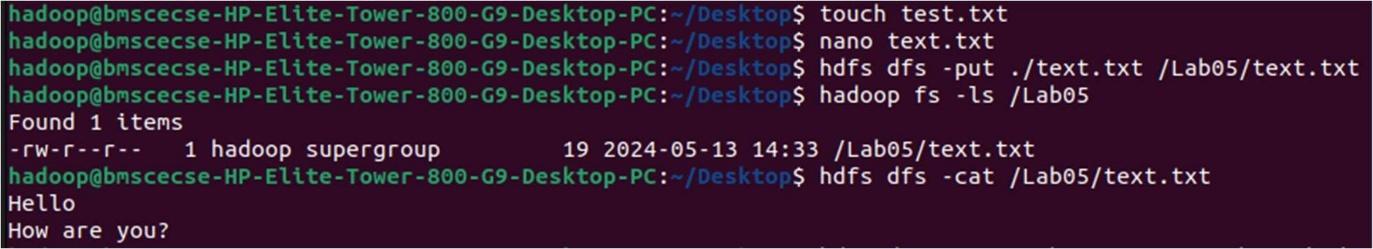
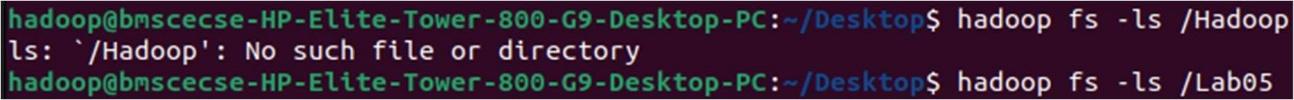


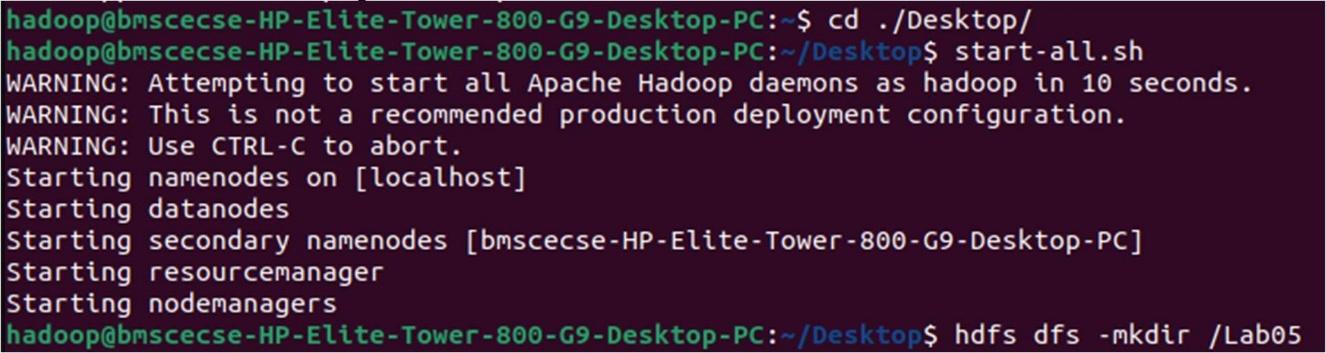


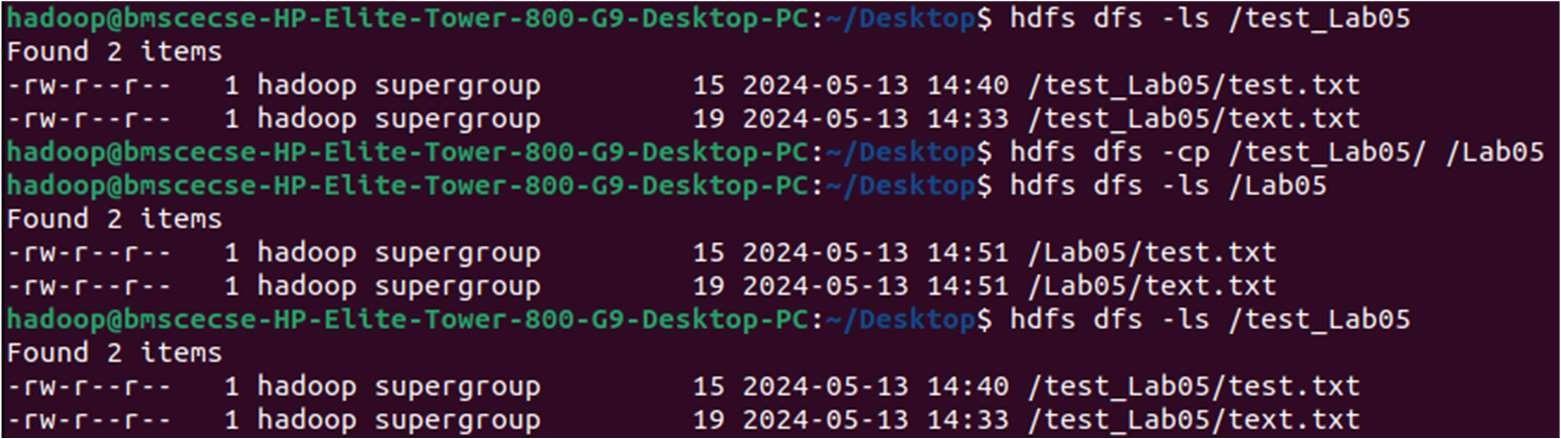
**Experiment - 4**

Q) Execution of HDFS Commands for interaction with Hadoop Environment. (Minimum 10 commands to be executed)

**Code & Output:**







**Experiment - 5**

Q) Implement Wordcount program on Hadoop framework

Code:

**Mapper Code: WCMapper.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.apache.hadoop.mapred.MapReduceBase;

import org.apache.hadoop.mapred.Mapper;

import org.apache.hadoop.mapred.OutputCollector;

import org.apache.hadoop.mapred.Reporter;

public class WCMapper extends MapReduceBase implements Mapper<LongWritable, Text, Text, IntWritable> {

public void map(LongWritable key, Text value, OutputCollector<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 Code: WCReducer.java**

import java.io.IOException;

import java.util.Iterator;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.MapReduceBase;

import org.apache.hadoop.mapred.OutputCollector;

import org.apache.hadoop.mapred.Reducer;

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));

}

}

**Driver Code: WCDriver.java**

import java.io.IOException;

import org.apache.hadoop.conf.Configured;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.FileInputFormat;

import org.apache.hadoop.mapred.FileOutputFormat;

import org.apache.hadoop.mapred.JobClient;

import org.apache.hadoop.mapred.JobConf;

import org.apache.hadoop.util.Tool;

import org.apache.hadoop.util.ToolRunner;

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.setOutputPath(conf, new Path(args[1]));

conf.setMapperClass(WCMapper.class);

conf.setReducerClass(WCReducer.class);

conf.setMapOutputKeyClass(Text.class);

conf.setMapOutputValueClass(IntWritable.class);

conf.setOutputKeyClass(Text.class);

conf.setOutputValueClass(IntWritable.class);

JobClient.runJob(conf);

return 0;

}

public static void main(String args[]) throws Exception {

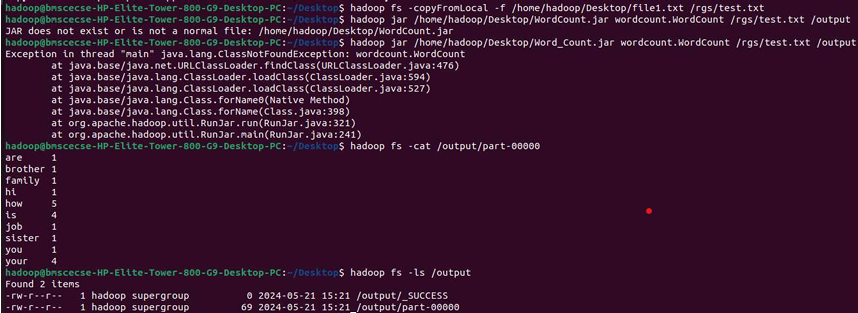
int exitCode = ToolRunner.run(new WCDriver(), args);

System.out.println(exitCode);

}

}

**Output:**



**Experiment – 6**

Q) From the following link extract the weather data  
<https://github.com/tomwhite/hadoopbook/tree/master/input/ncdc/all>

Create a Map Reduce program to  
a) find average temperature for each year from NCDC data set.  
b) find the mean max temperature for every month.

**Code:**

a) Find average temperature for each year from NCDC data set

**AverageDriver.java**

package temp;

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 AverageDriver {

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(AverageDriver.class);

job.setJobName("Max temperature");

FileInputFormat.addInputPath(job, new Path(args[0]));

FileOutputFormat.setOutputPath(job, new Path(args[1]));

job.setMapperClass(AverageMapper.class);

job.setReducerClass(AverageReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

System.exit(job.waitForCompletion(true) ? 0 : 1);

}

}

**AverageMapper.java**

package temp;

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 AverageMapper 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));

}

}

**AverageReducer.java**

package temp;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

public class AverageReducer 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++;

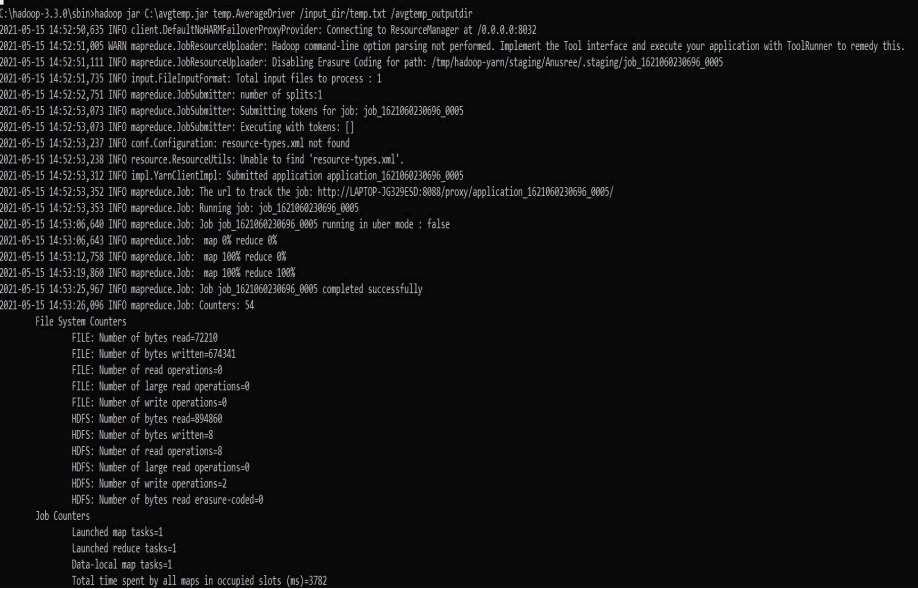
}

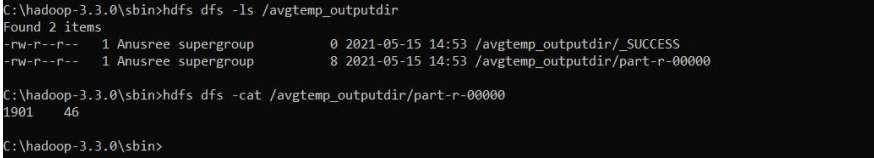
context.write(key, new IntWritable(max\_temp / count));

}

}

**Output:**





**b) Find the mean max temperature for every month**

**MeanMaxDriver.java**

package meanmax;

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 MeanMaxDriver {

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(MeanMaxDriver.class);

job.setJobName("Max temperature");

FileInputFormat.addInputPath(job, new Path(args[0]));

FileOutputFormat.setOutputPath(job, new Path(args[1]));

job.setMapperClass(MeanMaxMapper.class);

job.setReducerClass(MeanMaxReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

System.exit(job.waitForCompletion(true) ? 0 : 1);

}

}

**MeanMaxMapper.java**

java

CopyEdit

package meanmax;

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 MeanMaxMapper 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 month = line.substring(19, 21);

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(month), new IntWritable(temperature));

}

}

**MeanMaxReducer.java**

java

CopyEdit

package meanmax;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

public class MeanMaxReducer 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 total\_temp = 0;

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;

max\_temp = 0;

count = 0;

days++;

}

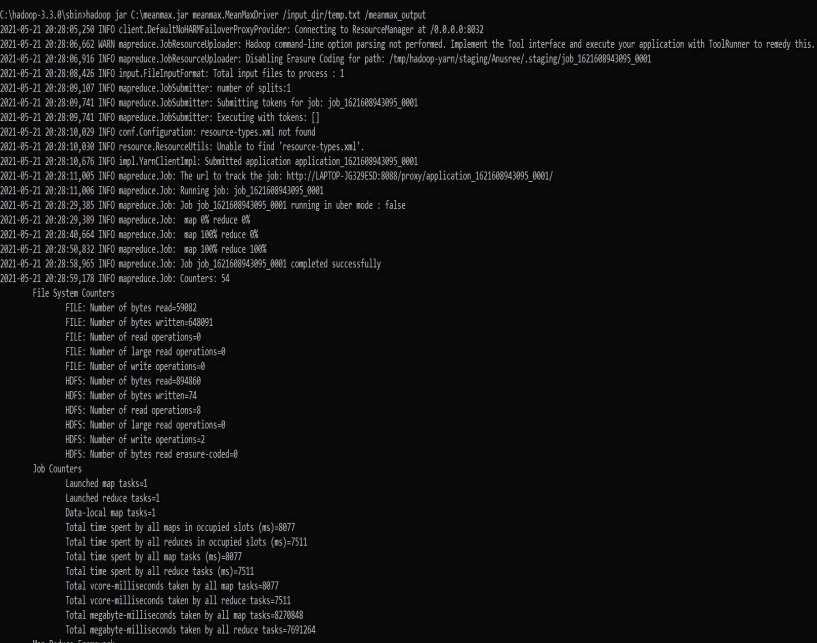
}

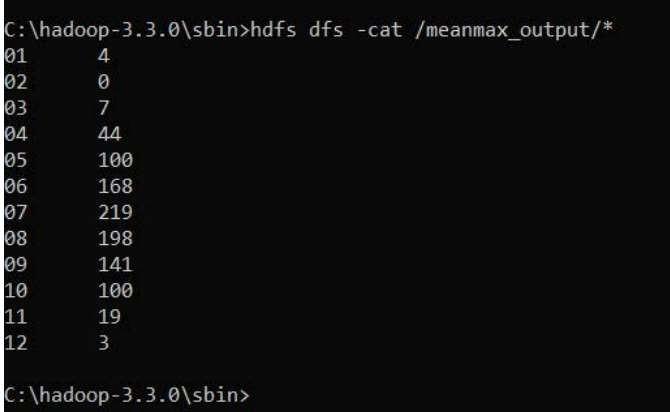
context.write(key, new IntWritable(total\_temp / days));

}

}

**Output:**

****



**Experiment – 7**

Q) For a given Text file, Create a Map Reduce program to sort the content in an alphabetic order listing only top 10 maximum occurrences of words.

Code :

**TopN.java (Driver)**

java

CopyEdit

package samples.topn;

import java.io.IOException;

import java.util.StringTokenizer;

import org.apache.hadoop.conf.Configuration;

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.Mapper;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;

import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

import org.apache.hadoop.util.GenericOptionsParser;

public class TopN {

public static void main(String[] args) throws Exception {

Configuration conf = new Configuration();

String[] otherArgs = (new GenericOptionsParser(conf, args)).getRemainingArgs();

if (otherArgs.length != 2) {

System.err.println("Usage: TopN <in> <out>");

System.exit(2);

}

Job job = Job.getInstance(conf);

job.setJobName("Top N");

job.setJarByClass(TopN.class);

job.setMapperClass(TopNMapper.class);

job.setReducerClass(TopNReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

FileInputFormat.addInputPath(job, new Path(otherArgs[0]));

FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));

System.exit(job.waitForCompletion(true) ? 0 : 1);

}

public static class TopNMapper extends Mapper<Object, Text, Text, IntWritable> {

private static final IntWritable one = new IntWritable(1);

private Text word = new Text();

private String tokens = "[\_|$#<>\\^=\\[\\]\\\*/\\\\,;,.\\-:()?!\"']";

public void map(Object key, Text value, Mapper<Object, Text, Text, IntWritable>.Context context)

throws IOException, InterruptedException {

String cleanLine = value.toString().toLowerCase().replaceAll(this.tokens, " ");

StringTokenizer itr = new StringTokenizer(cleanLine);

while (itr.hasMoreTokens()) {

this.word.set(itr.nextToken().trim());

context.write(this.word, one);

}

}

}

}

**TopNCombiner.java**

java

CopyEdit

package samples.topn;

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

public class TopNCombiner 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 sum = 0;

for (IntWritable val : values)

sum += val.get();

context.write(key, new IntWritable(sum));

}

}

**TopNMapper.java**

java

CopyEdit

package samples.topn;

import java.io.IOException;

import java.util.StringTokenizer;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Mapper;

public class TopNMapper extends Mapper<Object, Text, Text, IntWritable> {

private static final IntWritable one = new IntWritable(1);

private Text word = new Text();

private String tokens = "[\_|$#<>\\^=\\[\\]\\\*/\\\\,;,.\\-:()?!\"']";

public void map(Object key, Text value, Mapper<Object, Text, Text, IntWritable>.Context context)

throws IOException, InterruptedException {

String cleanLine = value.toString().toLowerCase().replaceAll(this.tokens, " ");

StringTokenizer itr = new StringTokenizer(cleanLine);

while (itr.hasMoreTokens()) {

this.word.set(itr.nextToken().trim());

context.write(this.word, one);

}

}

}

**TopNReducer.java**

java

CopyEdit

package samples.topn;

import java.io.IOException;

import java.util.HashMap;

import java.util.Map;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Reducer;

import utils.MiscUtils;

public class TopNReducer extends Reducer<Text, IntWritable, Text, IntWritable> {

private Map<Text, IntWritable> countMap = new HashMap<>();

public void reduce(Text key, Iterable<IntWritable> values,

Reducer<Text, IntWritable, Text, IntWritable>.Context context)

throws IOException, InterruptedException {

int sum = 0;

for (IntWritable val : values)

sum += val.get();

this.countMap.put(new Text(key), new IntWritable(sum));

}

protected void cleanup(Reducer<Text, IntWritable, Text, IntWritable>.Context context)

throws IOException, InterruptedException {

Map<Text, IntWritable> sortedMap = MiscUtils.sortByValues(this.countMap);

int counter = 0;

for (Text key : sortedMap.keySet()) {

if (counter++ == 20)

break;

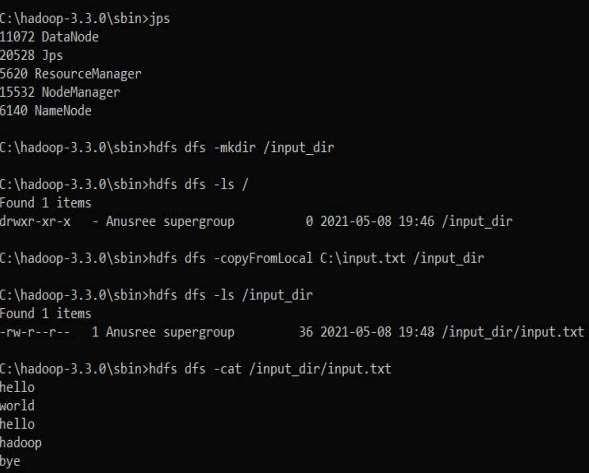
context.write(key, sortedMap.get(key));

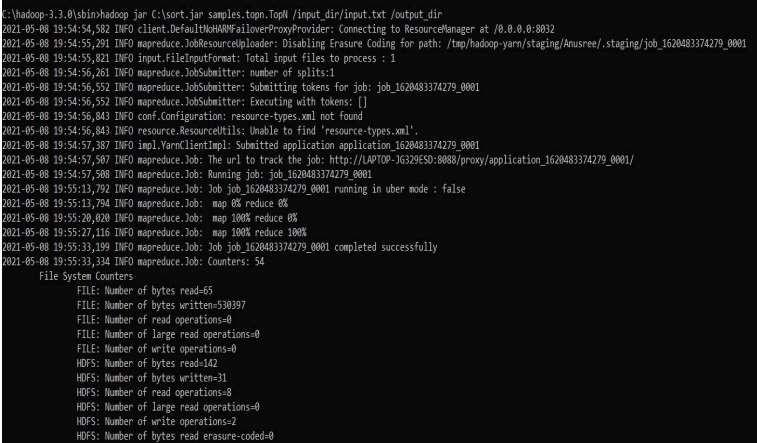
}

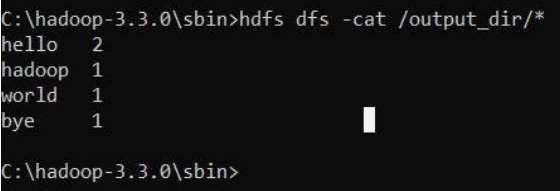
}

}

**Output:**







**Experiment – 8**

Q) Write a Scala program to print numbers from 1 to 100 using for loop.

**Code :** object ExampleForLoop1 {

def main(args: Array[String]): Unit = {

for (counter <- 1 to 100)

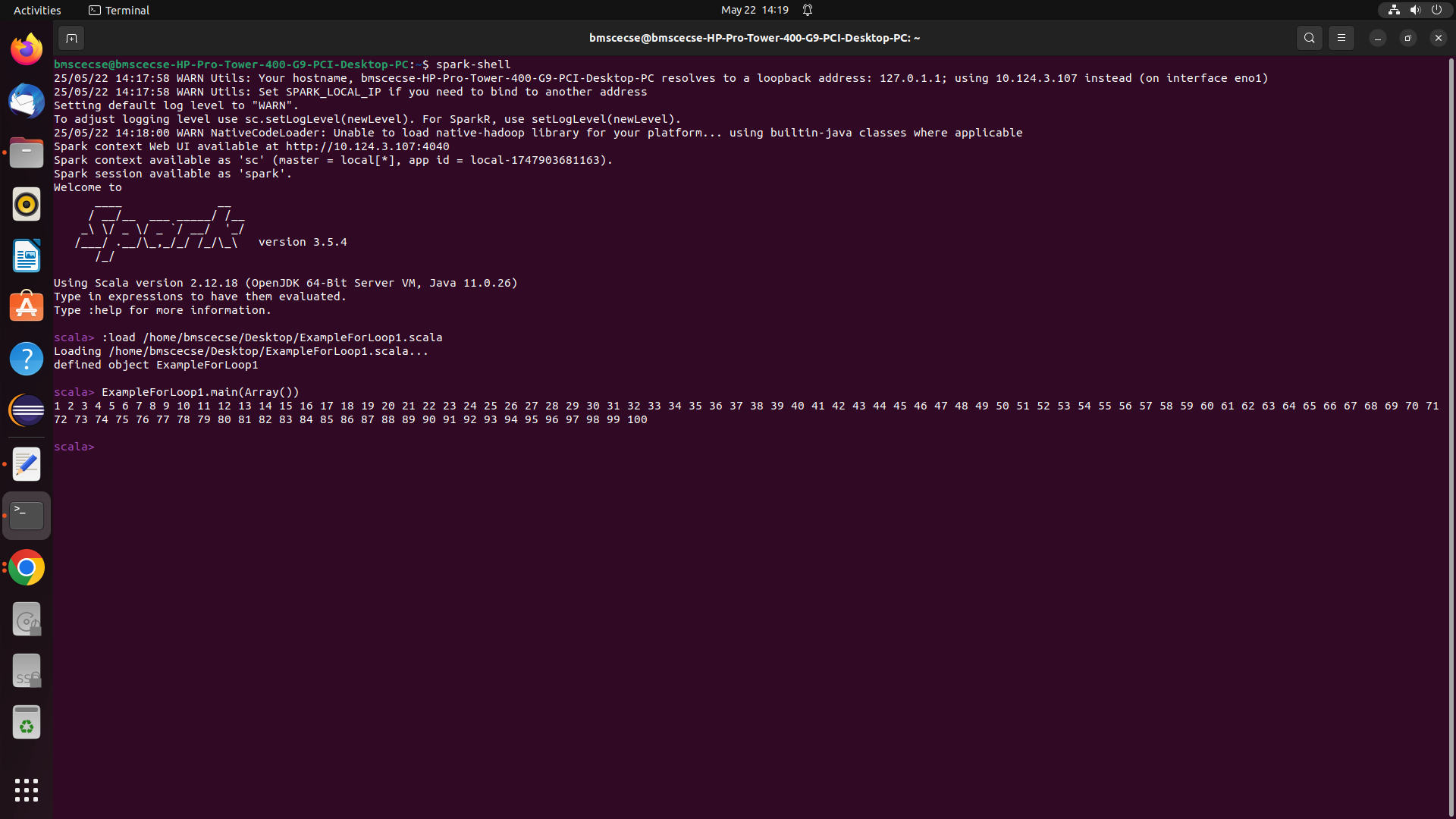
print(counter + " ")

// to print new line

println()

}

}

**Output:**  
 ****

**Experiment-9**

Q) 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

**Code :**

import org.apache.spark.sql.SparkSession

object FilterWordCount {

def main(args: Array[String]): Unit = {

if (args.length < 1) {

System.err.println("Usage: FilterWordCount <file>")

System.exit(1)

}

val spark = SparkSession.builder()

.appName("FilterWordCount")

.master("local[\*]") // Use local[\*] for running on local machine with multiple cores

.getOrCreate()

val rdd = spark.sparkContext.textFile(args(0))

val counts = rdd

.flatMap(\_.split("\\s+"))

.map(\_.replaceAll("""[\p{Punct}]""", ""))

.filter(\_.nonEmpty)

.map(w => (w.toLowerCase, 1))

.reduceByKey(\_ + \_)

.filter(\_.\_2 > 4)

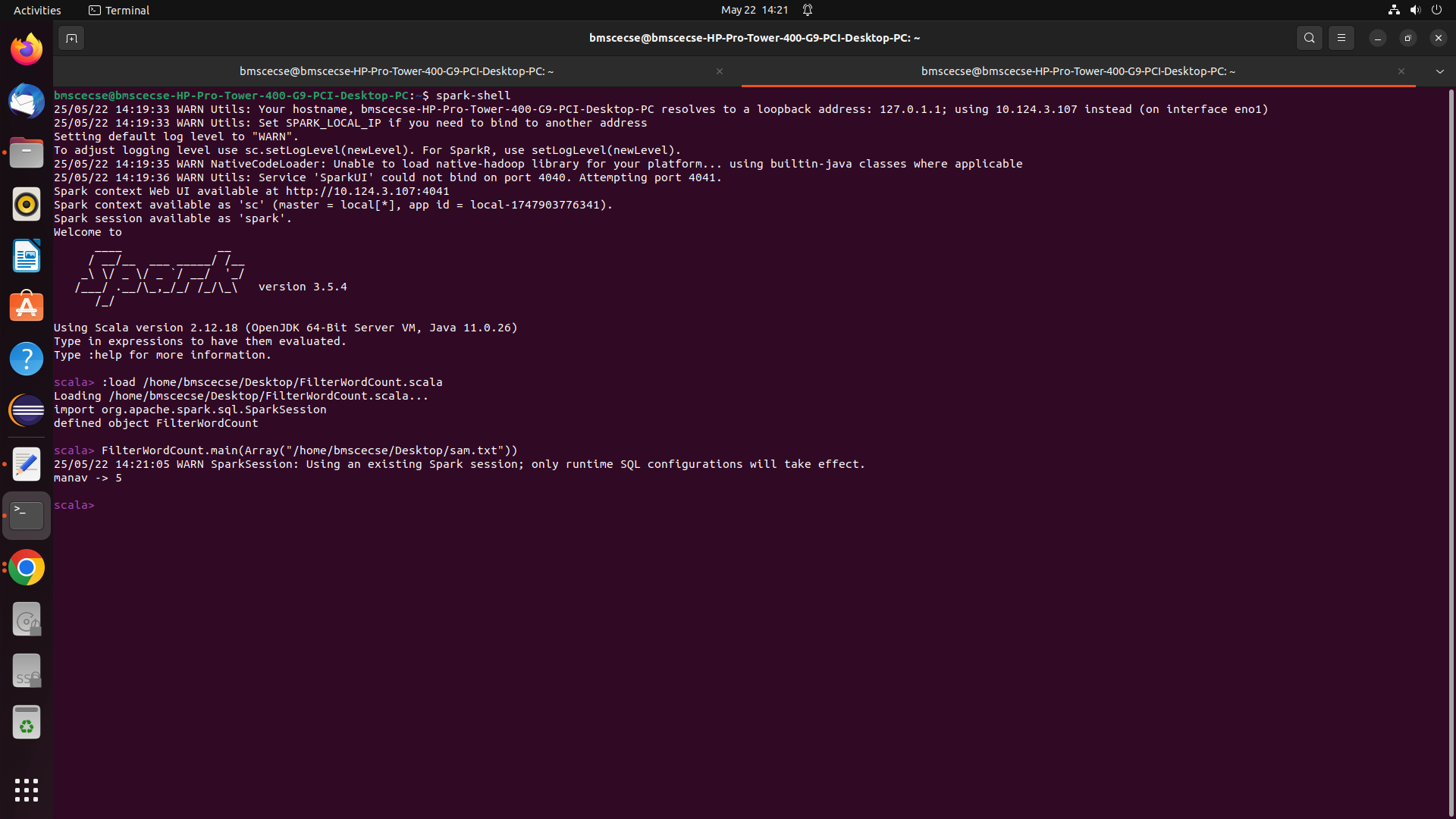
counts.collect().foreach{ case (w, c) => println(s"$w -> $c") }

spark.stop()

}

}

**Output**



**Experiment-10**

Q) Write a simple streaming program in Spark to receive text data streams on a particular port, perform basic text cleaning (like white space removal, stop words removal, lemmatization, etc.), and print the cleaned text on the screen. (Open Ended Question).

**Code:**  
import org.apache.spark.streaming.{Seconds, StreamingContext}

import org.apache.spark.ml.feature.{RegexTokenizer, StopWordsRemover}

import org.apache.spark.sql.functions.\_

object TextStreamCleaner {

def main(args: Array[String]): Unit = {

val ssc = new StreamingContext(sc, Seconds(5))

val sparkSession = spark

import sparkSession.implicits.\_

val lines = ssc.socketTextStream("localhost", 9999)\

lines.foreachRDD { rdd =>

if (!rdd.isEmpty()) {

val df = rdd.toDF("text")

val tokenizer = new RegexTokenizer()

.setInputCol("text")

.setOutputCol("words")

.setPattern("\\W")

val tokenizedDF = tokenizer.transform(df)

val remover = new StopWordsRemover()

.setInputCol("words")

.setOutputCol("filtered")

val cleanedDF = remover.transform(tokenizedDF)

cleanedDF.select("filtered").show(false)

}

}

ssc.start()

ssc.awaitTermination()

}

}

**Output**



