**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

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

****

**LAB REPORT**

**on**

**BIG DATA ANALYTICS**

**(20CS6PEBDA)**

***Submitted by***

**NEHA B CHADAGA (1BM19CS098)**

***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**

**May-2022 to July-2022**

**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 **NEHA B CHADAGA(1BM19CS098),** 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 a **Course Title - (Course code)** work prescribed for the said degree.

**Dr. Pallavi GB**              **Dr. Jyothi S Nayak**

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** | **4** |
| **2** | **Cassandra-Employee Database** | **8** |
| **3** | **Cassandra-Library Database** | **10** |
| **4** | **Hadoop Installation** | **13** |
| **5** | **HDFS Commands** | **14** |
| **6** | **Create a Map Reduce program to**   1. **find average temperature for each year from NCDC data set.** 2. **find the mean max temperature for every**   **month** | **16** |
| **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.** | **21** |
| **8** | **Create a Map Reduce program to demonstrate join operation** | **28** |
| **9** | **Program to print word count on scala shell**  **and print “Hello world” on scala IDE** | **35** |
| **10** | **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** | **37** |

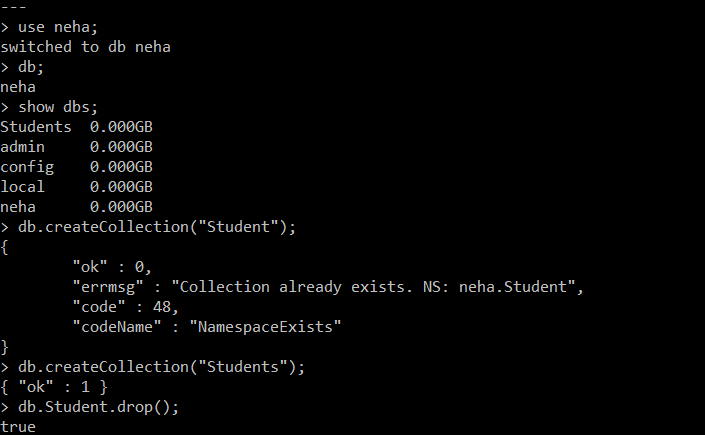
**Course Outcome**

|  |  |
| --- | --- |
| 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 |

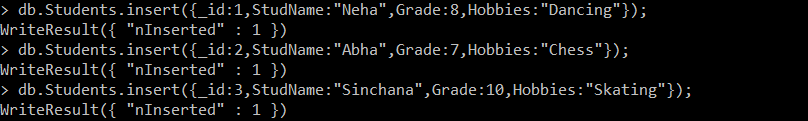
### Experiment 1

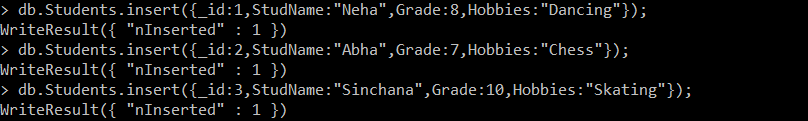
1

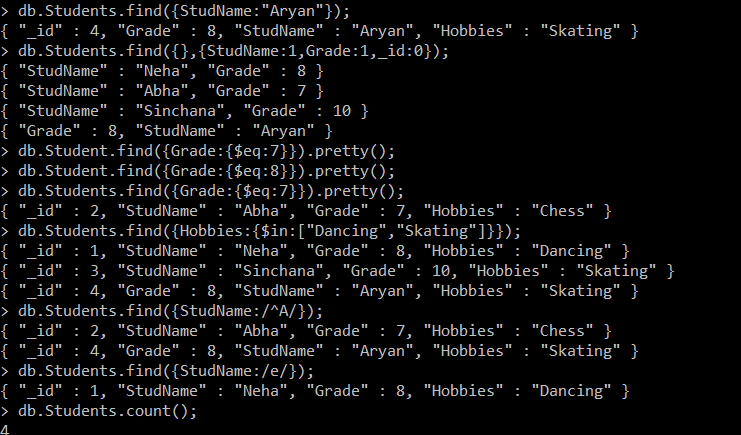
1. **CREATE DATABASE IN MONGODB.**

****

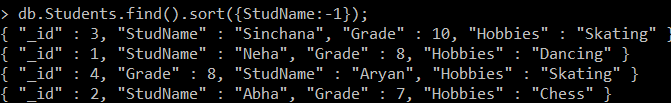
1. **CRUD (CREATE, READ, UPDATE, DELETE) OPERATIONS**

****

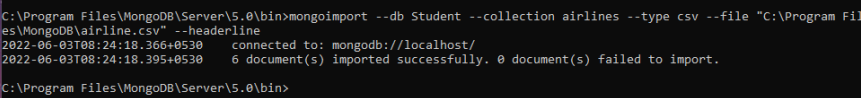




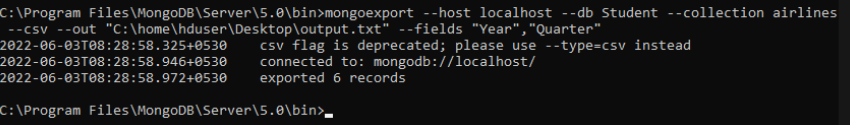
bda5



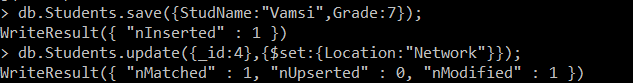
1. **Import data from a CSV file**

****

1. **Export data to a CSV file**

****

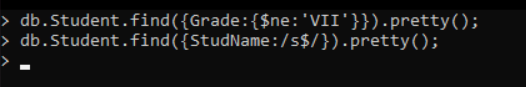
**V. Save Method and add a new field to existing Document:**

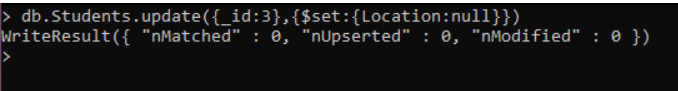


**VI. Other Operations**

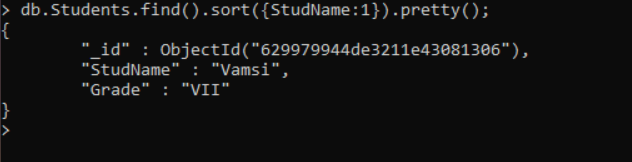
bda8

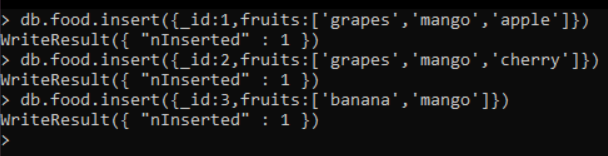
bda9

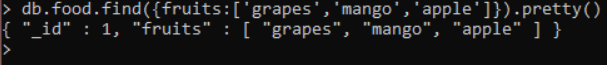




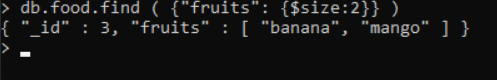
bda12



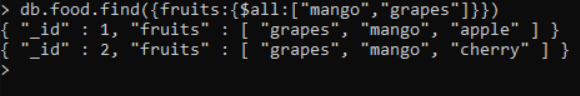


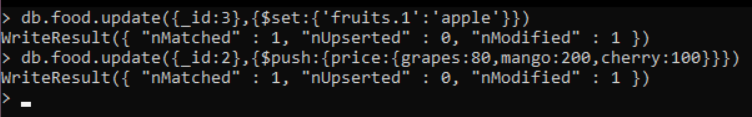


bda16

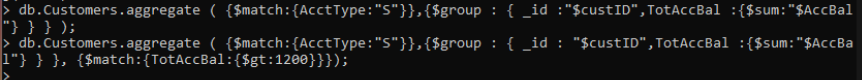


bda18

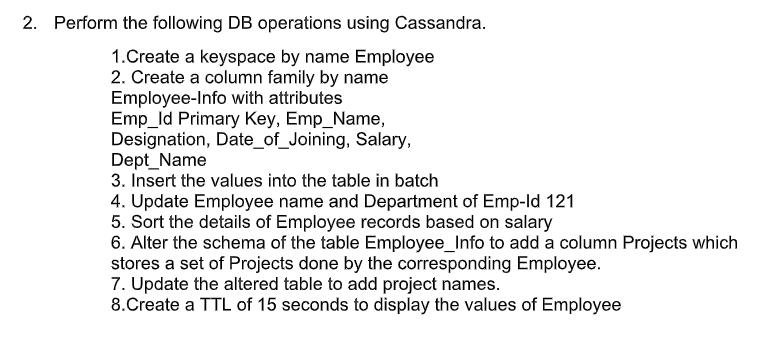


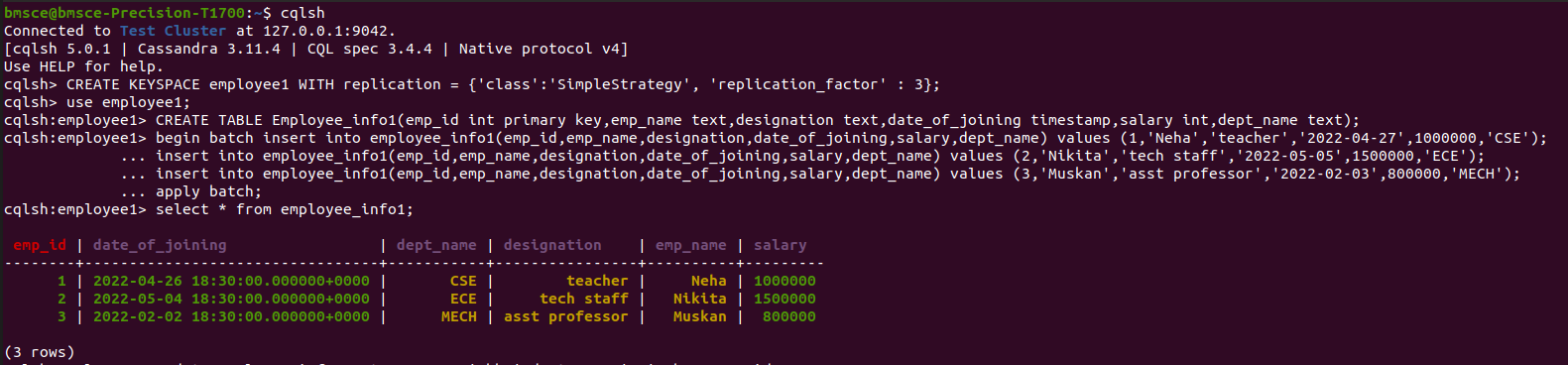


bda21

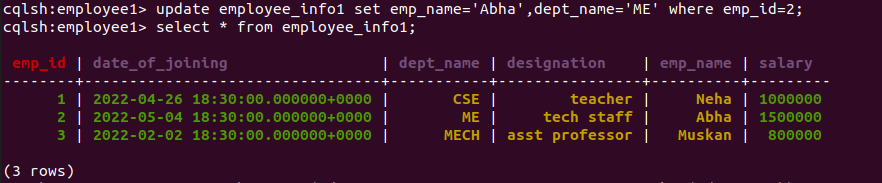


## EXPERIMENT 2

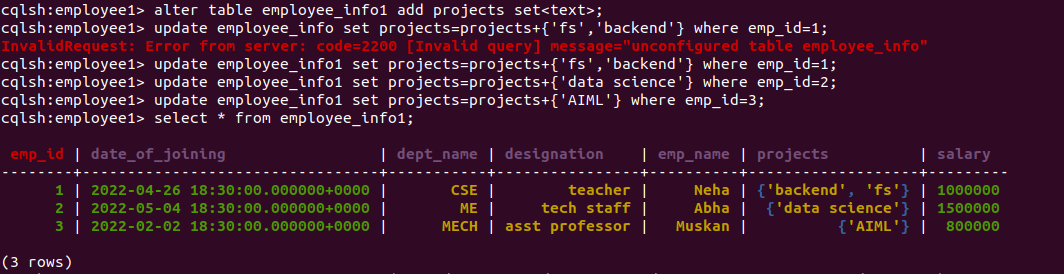


**1,2,3:**

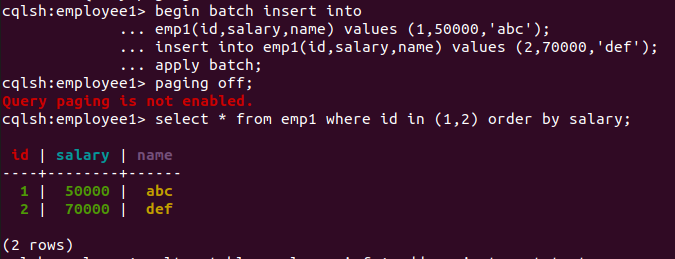
**4:**

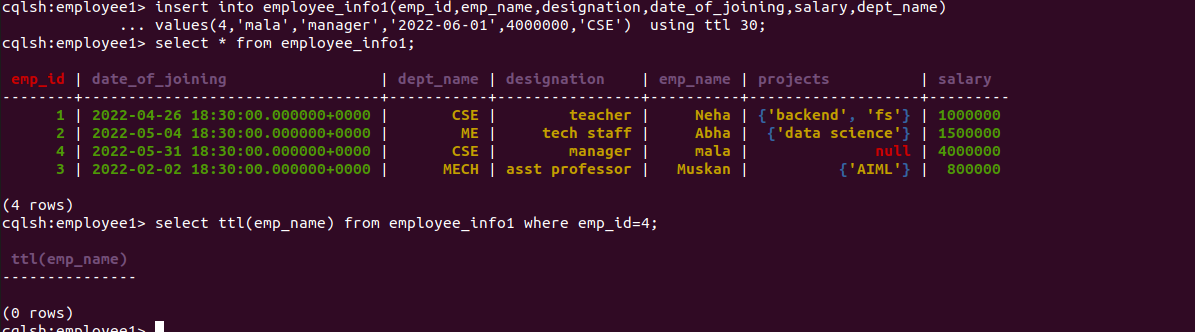
**5:**



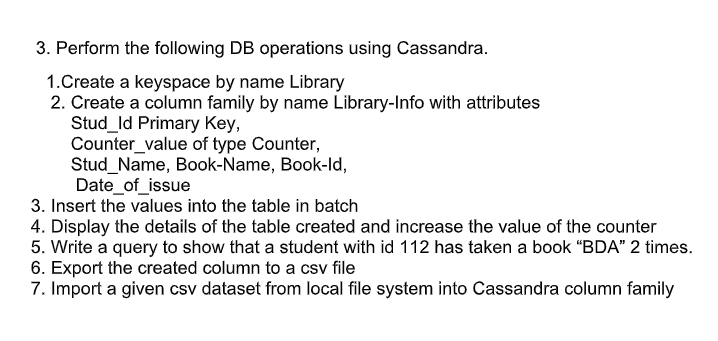


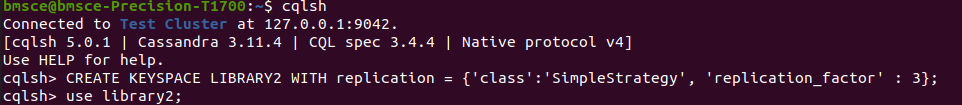
**6,7:**



**8:**

## EXPERIMENT 3

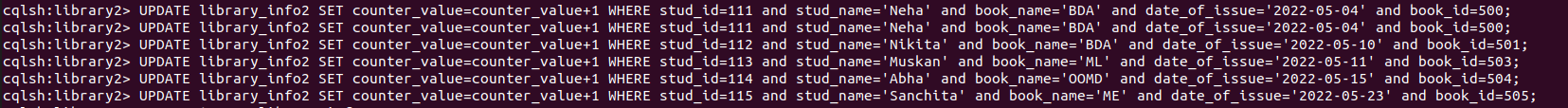


**1:**

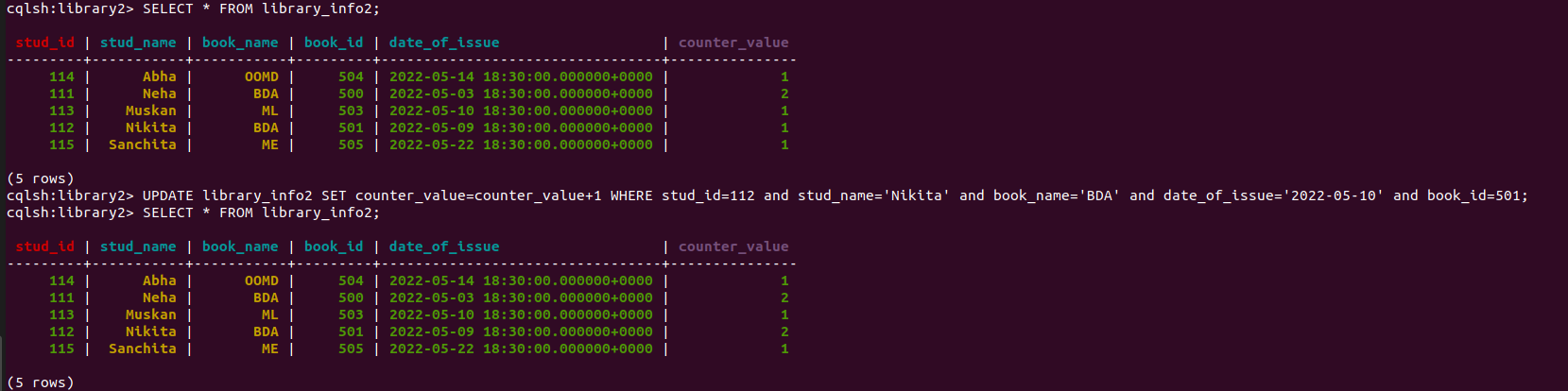
**2:**



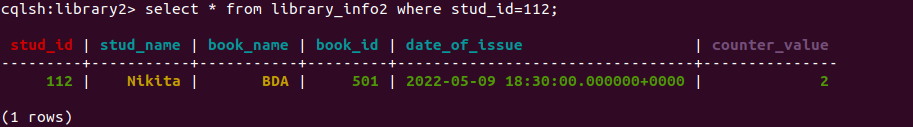
**3:**

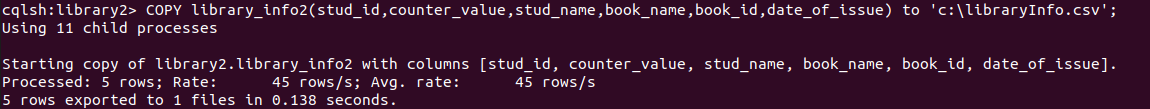


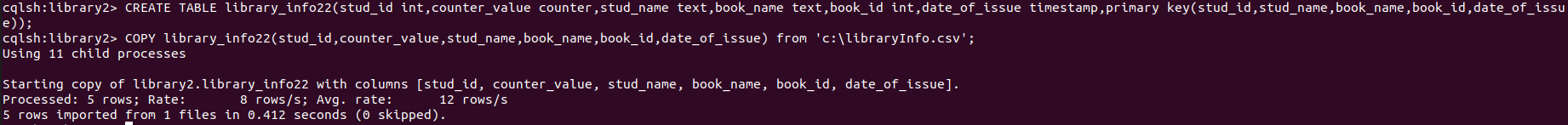
**4:**



**5:**

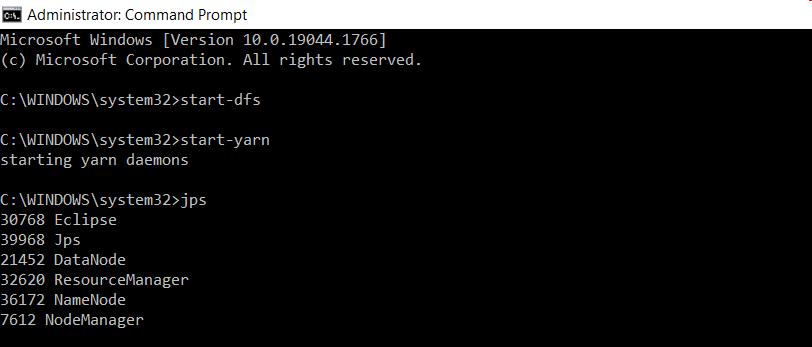


**6:**

**7:**

## Experiment 4

**Screenshot of Hadoop Installation:**

****

## Experiment 5

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

c:\hadoop\_new\sbin>hdfs dfs -mkdir /temp

c:\hadoop\_new\sbin>hdfs dfs -copyFromLocal E:\Desktop\sample.txt \temp

c:\hadoop\_new\sbin>hdfs dfs -ls \temp Found 1 items

-rw-r--r-- 1 Admin supergroup 11 2021-06-11 21:12 /temp/sample.txt

c:\hadoop\_new\sbin>hdfs dfs -cat \temp\sample.txt

hello world

c:\hadoop\_new\sbin>hdfs dfs -get \temp\sample.txt E:\Desktop\temp

c:\hadoop\_new\sbin>hdfs dfs -put E:\Desktop\temp \temp

c:\hadoop\_new\sbin>hdfs dfs -ls \temp Found 2 items

-rw-r--r-- 1 Admin supergroup 11 2021-06-11 21:12 /temp/sample.txt drwxr-xr-x -

Admin supergroup 0 2021-06-11 21:15 /temp/temp c:\hadoop\_new\sbin>hdfs dfs -mv \lab1 \temp

c:\hadoop\_new\sbin>hdfs dfs -ls \temp Found 3 items drwxr-xr-x - Admin supergroup 0 2021-04-19 15:07 /temp/lab1 -rw-r--r-- 1 Admin

supergroup 11 2021-06-11 21:12 /temp/sample.txt drwxr-xr-x -

Admin supergroup 0 2021-06-11 21:15 /temp/temp

c:\hadoop\_new\sbin>hdfs dfs -rm /temp/sample.txt Deleted /temp/sample.txt

c:\hadoop\_new\sbin>hdfs dfs -ls \temp Found 2 items drwxr-xr-x - Admin supergroup 0 2021-04-19 15:07 /temp/lab1 drwxr-xr-x - Admin

supergroup 0 2021-06-11 21:15 /temp/temp c:\hadoop\_new\sbin>hdfs dfs -copyFromLocal E:\Desktop\sample.txt \temp

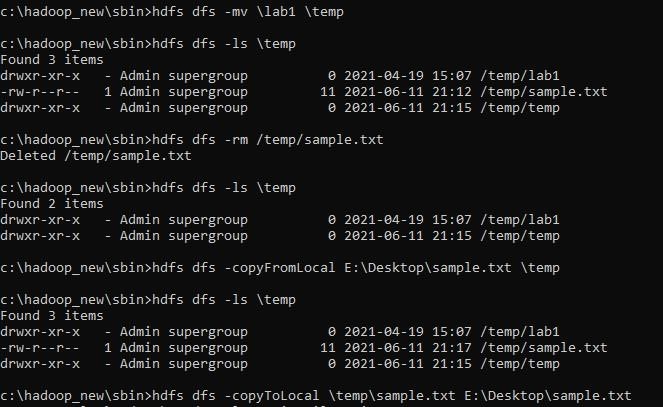
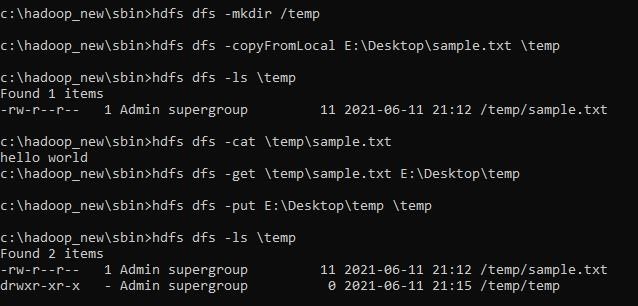
c:\hadoop\_new\sbin>hdfs dfs -ls \temp Found 3 items drwxr-xr-x - Admin supergroup 0 2021-04-19 15:07 /temp/lab1 -rw-r--r-- 1 Admin supergroup

11 2021-06-11 21:17 /temp/sample.txt drwxr-xr-x - Admin supergroup 0

2021-06-11 21:15 /temp/temp

c:\hadoop\_new\sbin>hdfs dfs -copyToLocal \temp\sample.txt E:\Desktop\sample.txt

**OUTPUT:**



## Experiment 6

**For the given file, Create a Map Reduce program to**

1. **Find the average temperature for each year from the NCDC data set.**

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

**CODE:**

a ) // AverageDriver.java package temperature;

import org.apache.hadoop.io.\*; import org.apache.hadoop.fs.\*; import org.apache.hadoop.mapreduce.\*; 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 temperature;

import org.apache.hadoop.io.\*; import org.apache.hadoop.mapreduce.\*; import java.io.IOException; public class AverageMapper extends Mapper <LongWritable, Text, Text, IntWritable>

{ public static final int MISSING = 9999;

public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException

{

String line = value.toString(); String year = line.substring(15,19); int temperature;

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 != MISSING && quality.matches("[01459]"))

context.write(new Text(year),new IntWritable(temperature)); }

}

//AverageReducer.java package temperature;

import org.apache.hadoop.io.IntWritable; import org.apache.hadoop.io.Text; import org.apache.hadoop.mapreduce.\*; import java.io.IOException;

public class AverageReducer extends Reducer <Text, IntWritable,Text, IntWritable>

{

public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException,InterruptedException

{

int max\_temp = 0; int count = 0; for (IntWritable value : values)

{

max\_temp += value.get(); count+=1;

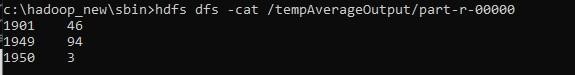
}

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

}

}

**OUTPUT:**



b ) //TempDriver.java package temperatureMax;

import org.apache.hadoop.io.\*; import org.apache.hadoop.fs.\*; import org.apache.hadoop.mapreduce.\*; import org.apache.hadoop.mapreduce.lib.input.FileInputFormat; import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class TempDriver

{ 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(TempDriver.class); job.setJobName("Max temperature");

FileInputFormat.addInputPath(job,new Path(args[0])); FileOutputFormat.setOutputPath(job,new Path (args[1]));

job.setMapperClass(TempMapper.class); job.setReducerClass(TempReducer.class);

job.setOutputKeyClass(Text.class); job.setOutputValueClass(IntWritable.class); System.exit(job.waitForCompletion(true)?0:1);

}

}

//TempMapper.java package temperatureMax;

import org.apache.hadoop.io.\*; import org.apache.hadoop.mapreduce.\*; import java.io.IOException;

public class TempMapper extends Mapper <LongWritable, Text, Text, IntWritable>

{ public static final int MISSING = 9999;

public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException

{

String line = value.toString(); String month = line.substring(19,21); int temperature; 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 != MISSING && quality.matches("[01459]")) context.write(new Text(month),new IntWritable(temperature)); }

}

//TempReducer.java package temperatureMax;

import org.apache.hadoop.io.\*; import org.apache.hadoop.mapreduce.\*; import java.io.IOException;

public class TempMapper extends Mapper <LongWritable, Text, Text, IntWritable>

{ public static final int MISSING = 9999;

public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException

{

String line = value.toString();

String month = line.substring(19,21); int temperature; if (line.charAt(87)=='+') temperature = Integer.parseInt(line.substring(88, 92));

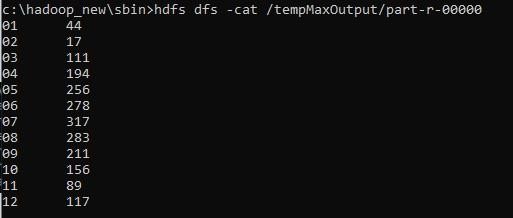
else

temperature = Integer.parseInt(line.substring(87, 92)); String quality = line.substring(92, 93); if(temperature != MISSING && quality.matches("[01459]")) context.write(new Text(month),new IntWritable(temperature));

}

}

**OUTPUT:**



## Experiment 7

**For a given Text file, create a Map Reduce program to sort the content in an alphabetic order listing only top ‘n’ maximum occurrence of words.**

**CODE:**

// TopN.java package sortWords;

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.Reducer;

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

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

import org.apache.hadoop.util.GenericOptionsParser; import utils.MiscUtils;

import java.io.IOException; import java.util.\*;

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.setCombinerClass(TopNReducer.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);

}

/\*\*

\* The mapper reads one line at the time, splits it into an array of single words and emits every \* word to the reducers with the value of 1.

\*/

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

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

private Text word = new Text();

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

@Override

public void map(Object key, Text value, Context context) throws IOException,

InterruptedException {

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

StringTokenizer itr= new StringTokenizer(cleanLine);

while (itr.hasMoreTokens()) { word.set(itr.nextToken().trim());

context.write(word, one);

}

}

}

/\*\*

\* The reducer retrieves every word and puts it into a Map: if the word already exists in the \* map, increments its value, otherwise sets it to 1.

\*/

public static class TopNReducer extends Reducer<Text, IntWritable, Text, IntWritable> { private Map<Text, IntWritable> countMap = new HashMap<>();

@Override

public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {

// computes the number of occurrences of a single word int sum = 0;

for (IntWritable val : values) {

sum += val.get();

}

// puts the number of occurrences of this word into the map.

// We need to create another Text object because the Text instance

// we receive is the same for all the words

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

}

@Override

protected void cleanup(Context context) throws IOException, InterruptedException {

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

break;

int counter = 0;

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

if (counter++ == 3) {

}

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

}

}

}

/\*\*

\* The combiner retrieves every word and puts it into a Map: if the word already exists in the \* map, increments its value, otherwise sets it to 1.

\*/

public static class TopNCombiner extends Reducer<Text, IntWritable, Text, IntWritable> {

@Override

public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {

// computes the number of occurrences of a single word

int sum = 0;

for (IntWritable val : values) {

sum += val.get();

}

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

}

}

}

// MiscUtils.java package utils; import java.util.\*;

public class MiscUtils {

/\*\*

sorts the map by values. Taken from:

http://javarevisited.blogspot.it/2012/12/how-to-sort-hashmap-java-by-key-and-value.html

\*/

public static <K extends Comparable, V extends Comparable> Map<K, V> sortByValues(Map<K, V> map) {

List<Map.Entry<K, V>> entries = new LinkedList<Map.Entry<K, V>>(map.entrySet());

Collections.sort(entries, new Comparator<Map.Entry<K, V>>() {

@Override

public int compare(Map.Entry<K, V> o1, Map.Entry<K, V> o2) {

return o2.getValue().compareTo(o1.getValue());

}

});

//LinkedHashMap will keep the keys in the order they are inserted

//which is currently sorted on natural ordering Map<K, V> sortedMap = new LinkedHashMap<K, V>();

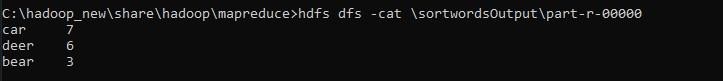
for (Map.Entry<K, V> entry : entries) { sortedMap.put(entry.getKey(), entry.getValue());

}

return sortedMap;

}

}

**OUTPUT:**

## Experiment 8

**Create a Hadoop Map Reduce program to combine information from the users file along with Information from the posts file by using the concept of join and display user\_id, Reputation and Score.**

**CODE:**

// JoinDriver.java

import org.apache.hadoop.conf.Configured;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.Text; import org.apache.hadoop.mapred.\*; import org.apache.hadoop.mapred.lib.MultipleInputs; import org.apache.hadoop.util.\*;

public class JoinDriver extends Configured implements Tool {

public static class KeyPartitioner implements Partitioner<TextPair, Text> { @Override

public void configure(JobConf job) {}

@Override

public int getPartition(TextPair key, Text value, int numPartitions) { return (key.getFirst().hashCode() & Integer.MAX\_VALUE) % numPartitions;

}

}

@Override public int run(String[] args) throws Exception { if (args.length != 3) { System.out.println("Usage: <Department Emp Strength input>

<Department Name input> <output>");

return -1;

}

JobConf conf = new JobConf(getConf(), getClass()); conf.setJobName("Join 'Department Emp Strength input' with 'Department Name input'");

Path AInputPath = new Path(args[0]); Path BInputPath = new Path(args[1]); Path outputPath = new Path(args[2]);

Posts.class); User.class);

MultipleInputs.addInputPath(conf, AInputPath, TextInputFormat.class, MultipleInputs.addInputPath(conf, BInputPath, TextInputFormat.class,

FileOutputFormat.setOutputPath(conf, outputPath);

conf.setPartitionerClass(KeyPartitioner.class); conf.setOutputValueGroupingComparator(TextPair.FirstComparator.class);

conf.setMapOutputKeyClass(TextPair.class);

conf.setReducerClass(JoinReducer.class); conf.setOutputKeyClass(Text.class);

JobClient.runJob(conf);

return 0;

}

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

int exitCode = ToolRunner.run(new JoinDriver(), args); System.exit(exitCode);

}

}

// JoinReducer.java import java.io.IOException; import java.util.Iterator; import org.apache.hadoop.io.Text; import org.apache.hadoop.mapred.\*;

public class JoinReducer extends MapReduceBase implements Reducer<TextPair, Text, Text, Text> {

@Override

public void reduce (TextPair key, Iterator<Text> values, OutputCollector<Text, Text> output, Reporter reporter)

throws IOException

{

Text nodeId = new Text(values.next()); while (values.hasNext()) { Text node = values.next();

Text outValue = new Text(nodeId.toString() + "\t\t" + node.toString()); output.collect(key.getFirst(), outValue);

}

}

}

// User.java import java.io.IOException; import java.util.Iterator; import org.apache.hadoop.conf.Configuration; import org.apache.hadoop.fs.FSDataInputStream; import org.apache.hadoop.fs.FSDataOutputStream; import org.apache.hadoop.fs.FileSystem; import org.apache.hadoop.fs.Path; import org.apache.hadoop.io.LongWritable; import org.apache.hadoop.io.Text; import org.apache.hadoop.mapred.\*;

import org.apache.hadoop.io.IntWritable;

public class User extends MapReduceBase implements Mapper<LongWritable, Text, TextPair, Text> { @Override

public void map(LongWritable key, Text value, OutputCollector<TextPair, Text> output, Reporter reporter)

throws IOException

{

String valueString = value.toString();

String[] SingleNodeData = valueString.split("\t"); output.collect(new TextPair(SingleNodeData[0], "1"), new

Text(SingleNodeData[1]));

}

}

//Posts.java

import java.io.IOException;

import org.apache.hadoop.io.\*; import org.apache.hadoop.mapred.\*;

public class Posts extends MapReduceBase implements Mapper<LongWritable, Text, TextPair, Text> { @Override

public void map(LongWritable key, Text value, OutputCollector<TextPair, Text> output, Reporter reporter)

throws IOException

{

String valueString = value.toString();

String[] SingleNodeData = valueString.split("\t"); output.collect(new

TextPair(SingleNodeData[3], "0"), new Text(SingleNodeData[9]));

}

}

// TextPair.java import java.io.\*; import org.apache.hadoop.io.\*;

public class TextPair implements WritableComparable<TextPair> { private Text first; private Text second;

public TextPair() { set(new Text(), new Text());

}

public TextPair(String first, String second) { set(new Text(first), new Text(second));

}

public TextPair(Text first, Text second) { set(first, second);

}

public void set(Text first, Text second) { this.first = first; this.second = second;

}

public Text getFirst() { return first;

}

public Text getSecond() { return second;

}

@Override

public void write(DataOutput out) throws IOException { first.write(out); second.write(out);

}

@Override public void readFields(DataInput in) throws IOException { first.readFields(in); second.readFields(in);

}

@Override public int hashCode() { return first.hashCode() \* 163 + second.hashCode();

}

@Override public boolean equals(Object o) { if (o instanceof TextPair) { TextPair tp = (TextPair) o; return first.equals(tp.first) && second.equals(tp.second);

} return false;

}

@Override public String toString() { return first + "\t" + second;

}

@Override

public int compareTo(TextPair tp) { int cmp = first.compareTo(tp.first); if (cmp != 0) { return cmp;

}

return second.compareTo(tp.second);

}

// ^^ TextPair

// vv TextPairComparator public static class Comparator extends WritableComparator { private static final Text.Comparator TEXT\_COMPARATOR = new Text.Comparator();

public Comparator() { super(TextPair.class);

}

@Override public int compare(byte[] b1, int s1, int l1, byte[] b2, int s2, int l2) { try {

int firstL1 = WritableUtils.decodeVIntSize(b1[s1]) + readVInt(b1, s1); int firstL2 = WritableUtils.decodeVIntSize(b2[s2]) + readVInt(b2, s2); int cmp = TEXT\_COMPARATOR.compare(b1, s1, firstL1, b2, s2, firstL2); if (cmp != 0) { return cmp;

}

return TEXT\_COMPARATOR.compare(b1, s1 + firstL1, l1 - firstL1,

b2, s2 + firstL2, l2 - firstL2);

} catch (IOException e) { throw new IllegalArgumentException(e);

}

}

}

static {

WritableComparator.define(TextPair.class, new Comparator());

}

public static class FirstComparator extends WritableComparator {

private static final Text.Comparator TEXT\_COMPARATOR = new Text.Comparator(); public FirstComparator() { super(TextPair.class);

}

@Override public int compare(byte[] b1, int s1, int l1, byte[] b2, int s2, int l2) { try {

int firstL1 = WritableUtils.decodeVIntSize(b1[s1]) + readVInt(b1, s1); int firstL2 = WritableUtils.decodeVIntSize(b2[s2]) + readVInt(b2, s2); return TEXT\_COMPARATOR.compare(b1, s1, firstL1, b2, s2, firstL2);

} catch (IOException e) { throw new IllegalArgumentException(e);

}

}

@Override

public int compare(WritableComparable a, WritableComparable b) { if (a instanceof TextPair && b instanceof TextPair) { return ((TextPair) a).first.compareTo(((TextPair) b).first);

}

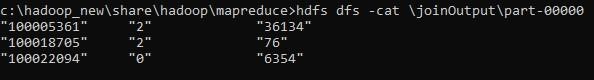
return super.compare(a, b);

}

}

}

**OUTPUT:**



## Experiment 9

**Program to print word count on scala shell and print “Hello world” on scala IDE.**

**CODE:**

scala> println("Hello World!");

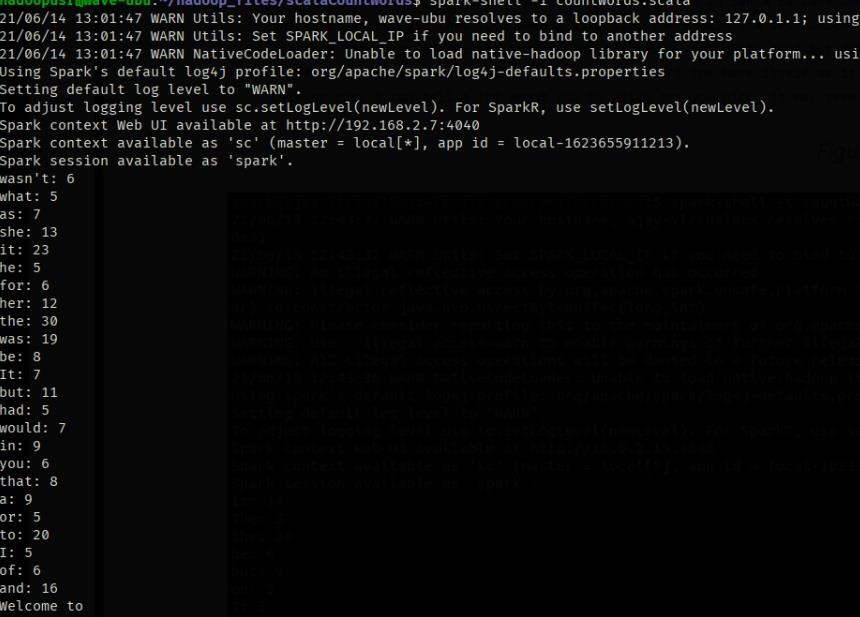
Hello World!

val data=sc.textFile("sparkdata.txt") data.collect;

val splitdata = data.flatMap(line => line.split(" ")); splitdata.collect;

val mapdata = splitdata.map(word => (word,1)); mapdata.collect;

val reducedata = mapdata.reduceByKey(\_+\_); reducedata.collect;

**OUTPUT:**

## Experiment 10

**Using RDD and Flat Map 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:**

