**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

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

****

**LAB REPORT**

**on**

**BIG DATA ANALYTICS**

**(29CS5PEBDA)**

***Submitted by***

**ASHWINI**

**(1BM20CS402)**

***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**

**APRIL-2022 to AUGUST-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 **ASHWINI (1BM20CS402),** 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 **BIG DATA ANALYTICS - (20CS6PEBDA)** work prescribed for the said degree.

Pallavi GB               **Dr. Jyothi S Nayak**

Assistant Professor Professor and Head

Department of CSE Department of CSE

BMSCE, Bengaluru BMSCE, Bengaluru

`

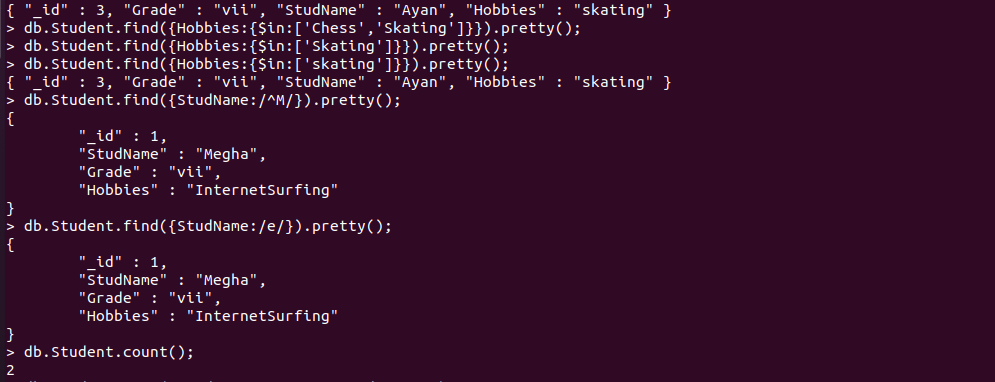
**Index Sheet**

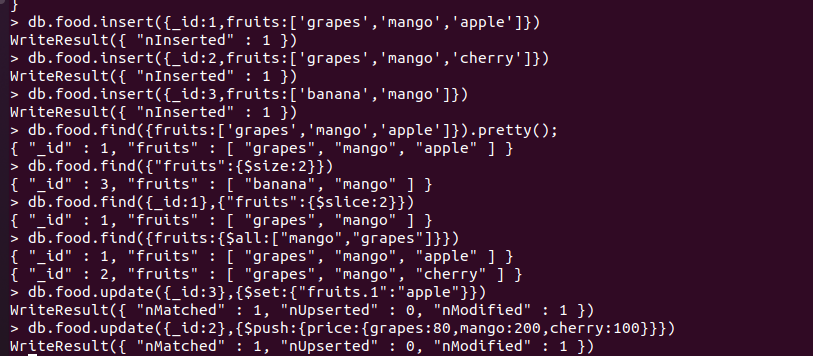
|  |  |  |
| --- | --- | --- |
| **Sl. No.** | **Experiment Title** | **Page No.** |
| **1** | **MongoDB CRUD Demonstration** | **4-5** |
| **2** | **Cassandra Employee Database** | **6-7** |
| **3** | **Cassandra Library Database** | **8-9** |
| **4** | **Screen shots of Hadoop Installation** | **10** |
| **5** | **Execution of HDFS Commands for interaction with Hadoop Environment.** | **11-14** |
| **6** | **Map Reduce program to a) find average temperature for each year from NCDC data  b) find the mean max temperature for every month.** | **15-17** |
| **7** | **Create a Map Reduce program to sort the content in an alphabetic order listing only top 10 maximum occurrences of words.** | **18-23** |
| **8** | **Create a Map Reduce program to demonstrating join operation.** | **24-29** |
| **9** | **Program to print word count on Scala shell and print "Hello World" on Scala IDE.** | **30-31** |
| **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.** | **32-33** |

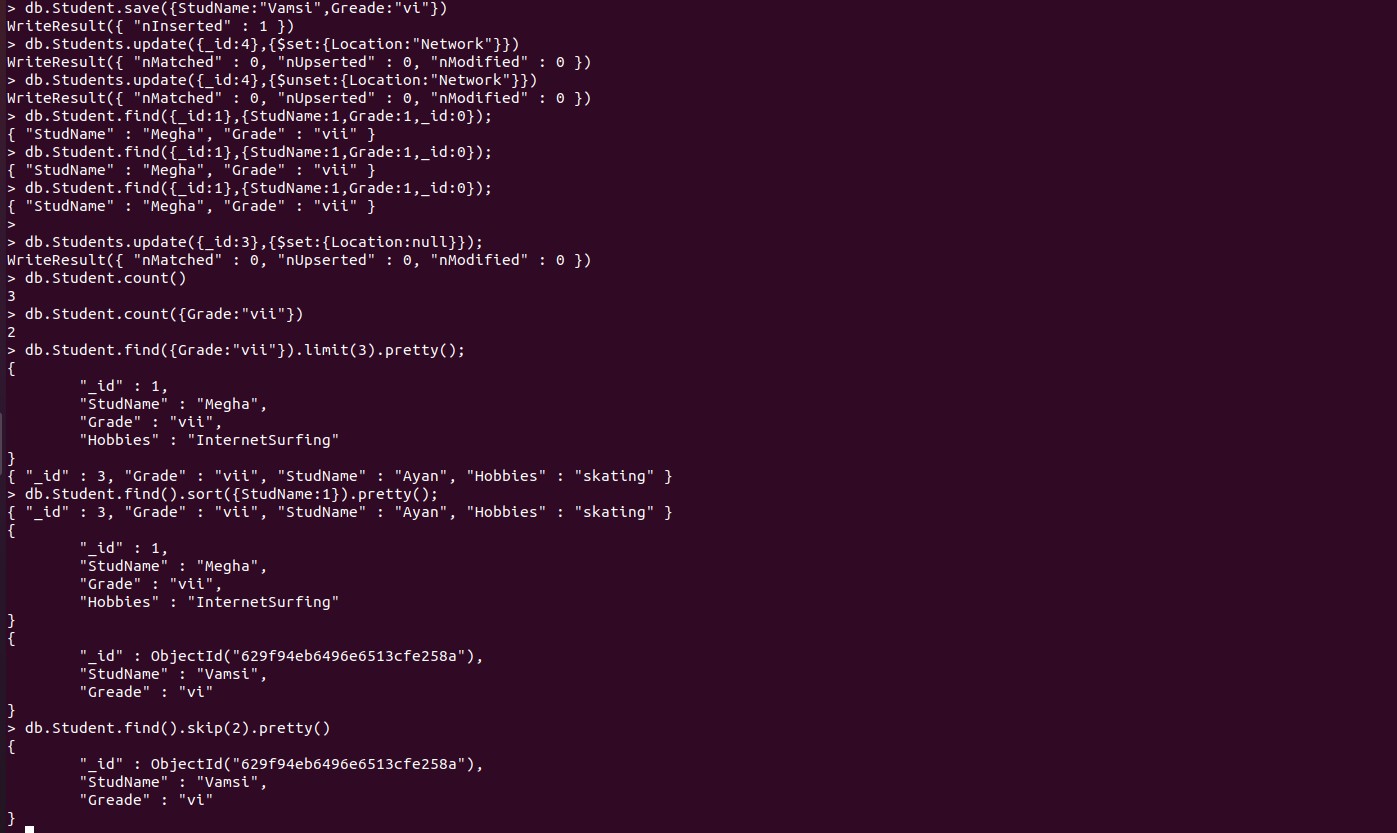
**Course Outcomes**

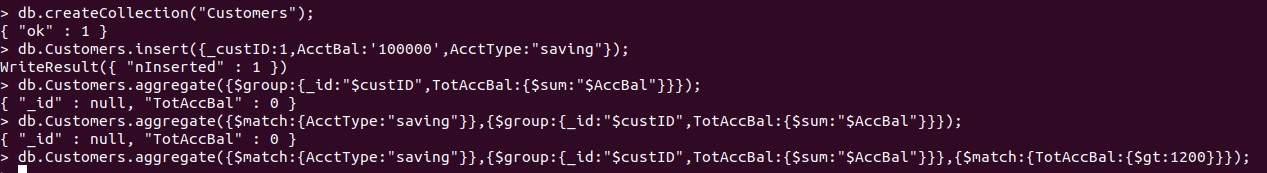
|  |  |
| --- | --- |
| **Co1** | Apply the concept of NoSQL, Hadoop or Spark for a given task |
| **C02** | Analyze the Big Data and obtain insight using data analytics mechanisms. |
| **C03** | Design and implement Big data applications by applying NoSQL, Hadoop or Spark |

1. **MongoDB CRUD Demonstration**

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

**Save method**

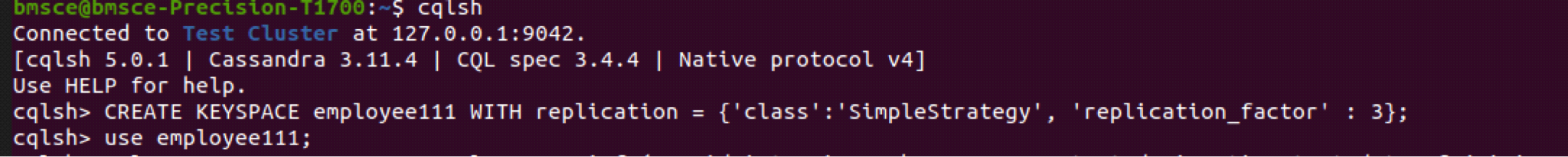


**Aggregate function**

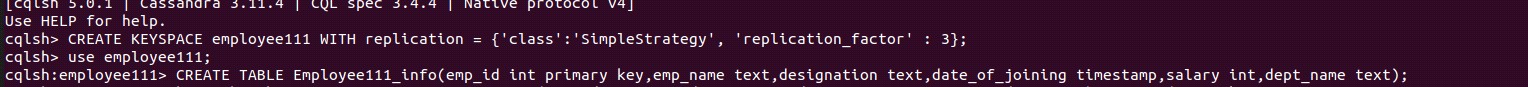
1. **Perform the following DB operations using Cassandra.**

**Create a keyspace by name Employee.**

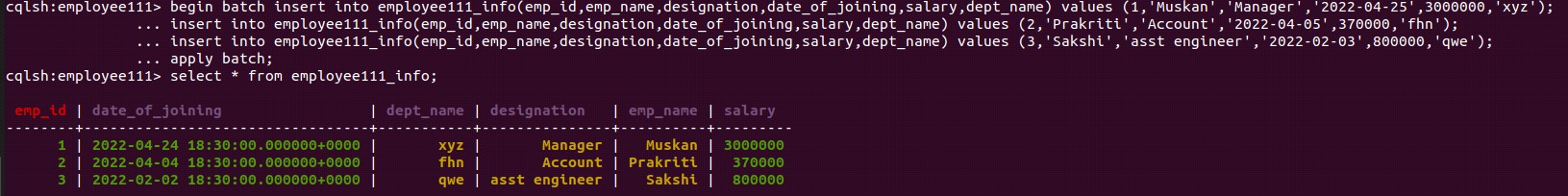
1. Create a key space by name Employee



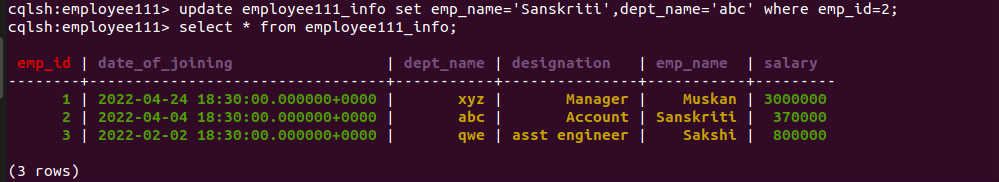
1. Create a column family by name Employee-Info with attributes Emp\_Id Primary Key, Emp\_Name, Designation, Date\_of\_Joining, Salary, Dept\_Name



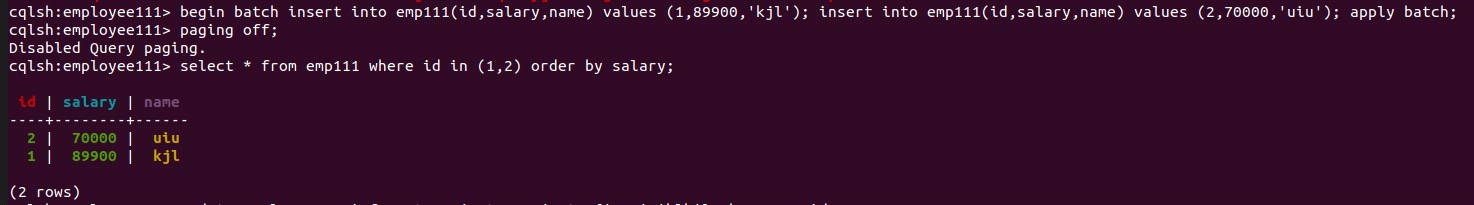
1. Insert the values into the table in batch



1. Update Employee name and Department of Emp-Id 2



1. Sort the details of Employee records based on salary

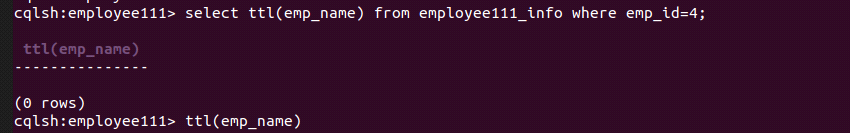
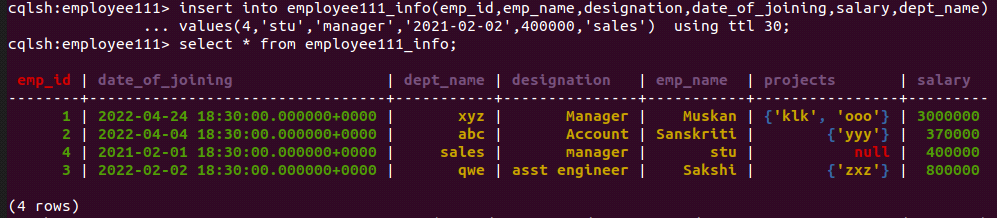
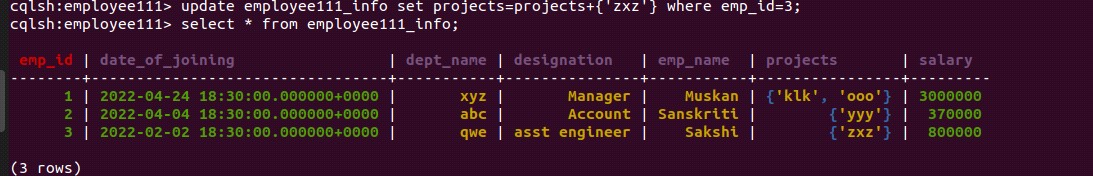




1. Alter the schema of the table Employee\_Info to add a column Projects which stores a set of Projects done by the corresponding Employee.



1. Update the altered table to add project names.



# **3. Perform the following DB operations using Cassandra.**

### Create a key space by name Library

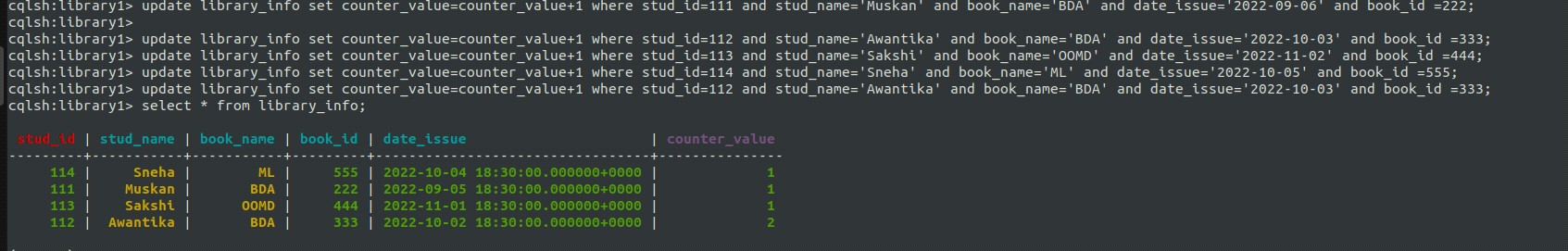




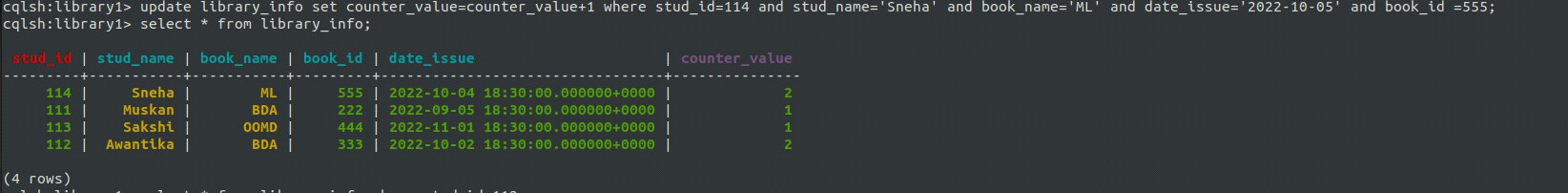
1. Create a column family by name Library-Info with attributes Stud\_Id Primary Key, Counter\_value of type Counter, Stud\_Name, Book-Name, Book-Id, Date\_of\_issue



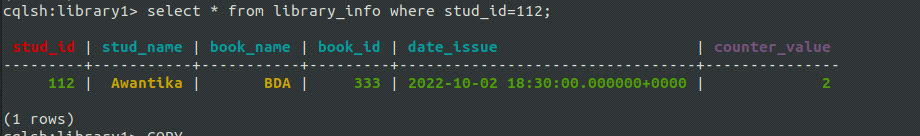
1. Insert the values into the table in batch



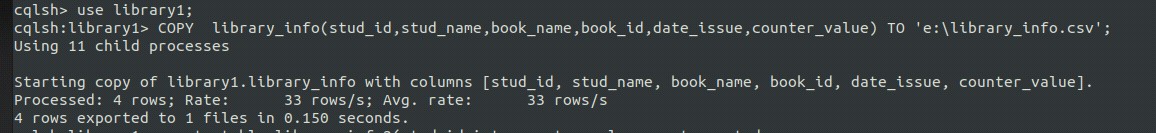
### Display the details of the table created and increase the value of the counter



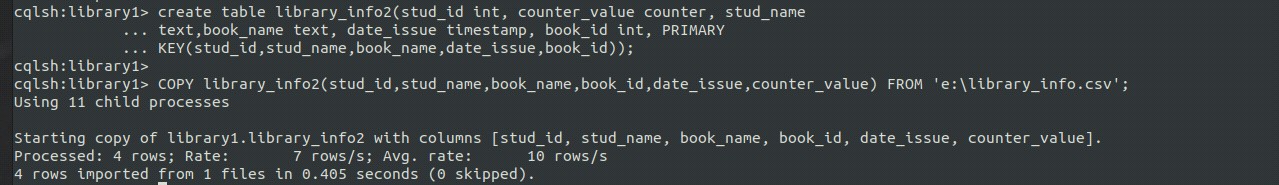
1. Write a query to show that a student with id 112 has taken a book “BDA” 2 times.



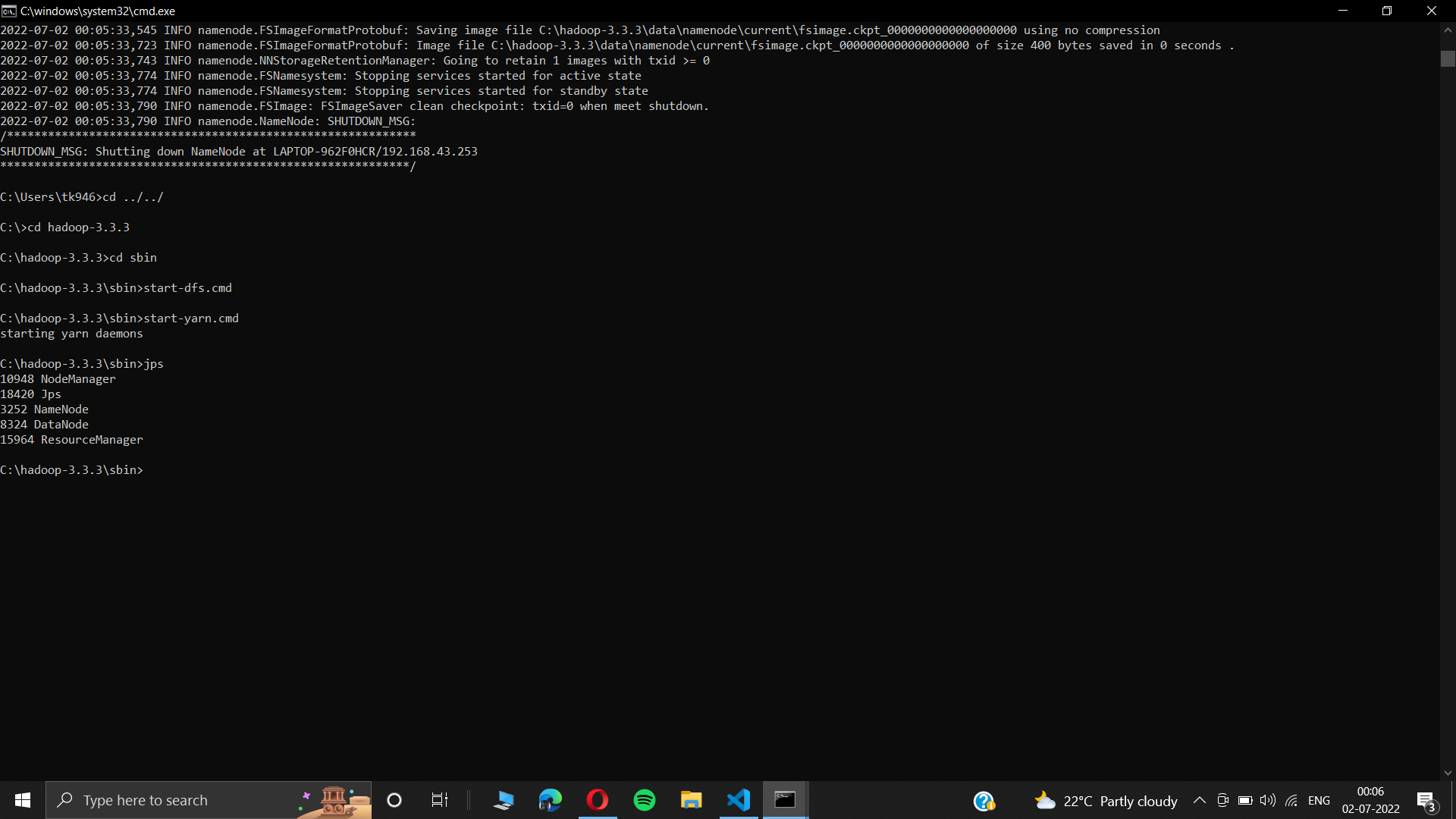
1. Export the created column to a csv file

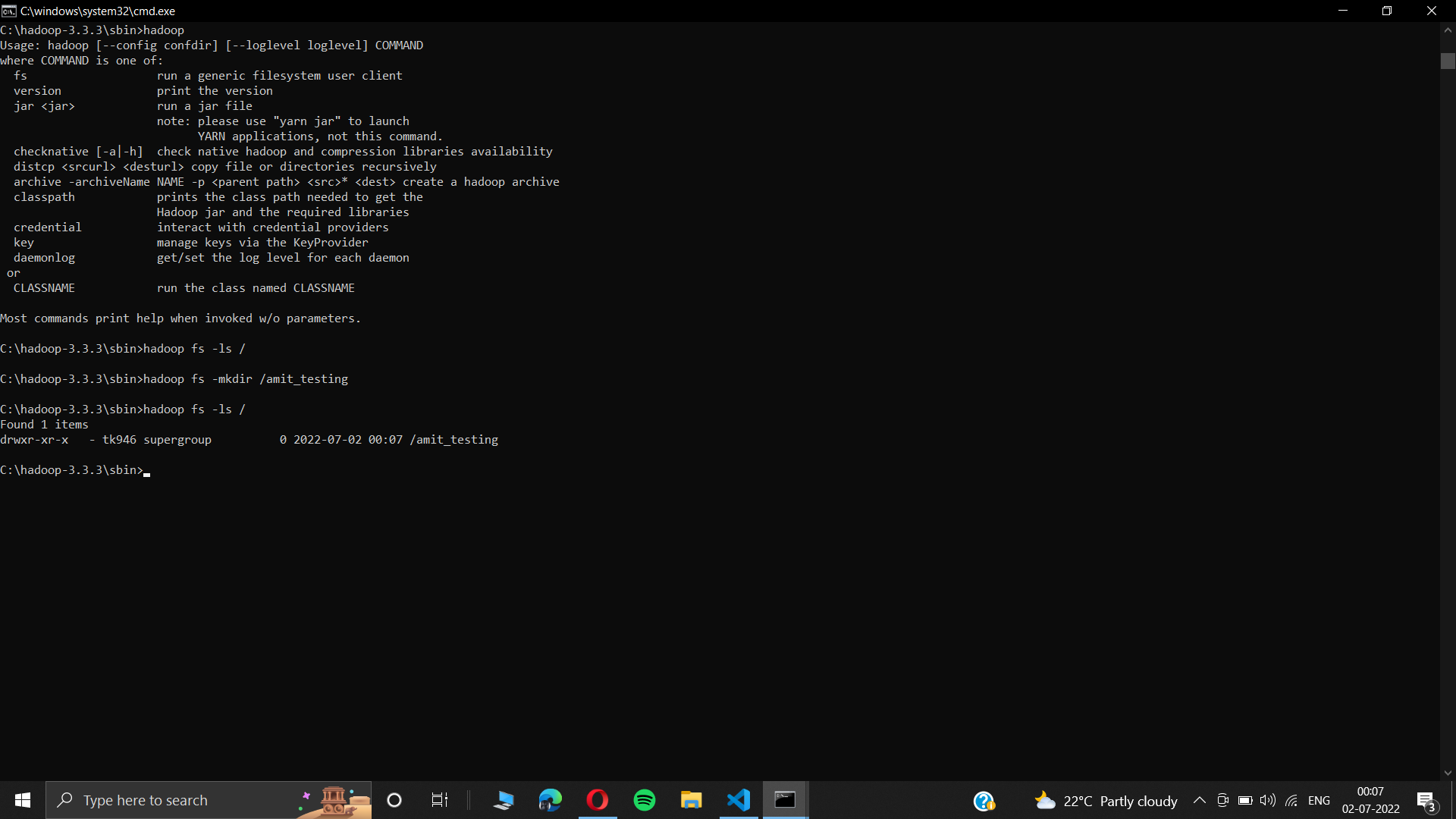


1. Import a given csv dataset from local file system into Cassandra column family



**4. ScreenShots of Hadoop installations**





## **5. Execution of HDFS Commands for interaction with Hadoop Environment.**

# 1. mkdir

Hadoop HDFS mkdir Command Usage  
mkdir  
Hadoop HDFS mkdir Command Example  
hdfs dfs -mkdir /abc  
Hadoop HDFS mkdir Command Description  
This HDFS command takes path URI’s as an argument and creates directories.

# 2. ls

Hadoop HDFS ls Command Usage  
ls  
Hadoop HDFS ls Command Example  
hadoop fs -ls /  
Hadoop HDFS ls Commnad Description  
This Hadoop HDFS ls command displays a list of the contents of a directory specified by path provided by the user, showing the names, permissions, owner, size and  
modification date for each entry.

# 3. put

Hadoop HDFS put Command Usage  
put  
Hadoop HDFS put Command Example  
hdfs dfs -put /home/hduser/Desktop/Welcome.txt /abc/WC.txt  
Hadoop  
HDFS put Command Description  
This hadoop basic command copies the file or directory from the local file system to the destination within the DFS. Display the contents of the file WC.txt hdfs dfs -cat /abc/WC.txt

# 4. copyFromLocal

Hadoop HDFS copyFromLocal Command Usage  
copyFromLocal  
Hadoop HDFS copyFromLocal Command Example  
hdfs dfs -put /home/hduser/Desktop/Welcome.txt /abc/WC.txt  
Hadoop HDFS copyFromLocal Command Description  
This hadoop shell command is similar to put command, but the source is restricted to a local file reference.  
Display the contents of the file WC2.txt hdfs dfs -cat /abc/WC2.txt

# 5. get

Hadoop HDFS get Command Usage  
get [-crc]

i. Hadoop HDFS get Command Example  
hdfs dfs -get /abc/WC.txt /home/hduser/Downloads/WWC.txt  
  
This HDFS fs command copies the file or directory in HDFS identified by the source to the local file system path identified by local destination.

ii. Hadoop HDFS get Command Example hdfs dfs -getmerge /abc/WC.txt /abc/WC2.txt /home/hduser/Desktop/Merge.txt

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

iii. Hadoop HDFS get Command Example hadoop fs -getfacl /abc/

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

# 6. copyToLocal

Hadoop HDFS copyToLocal Command Usage  
copyToLocal  
Hadoop HDFS copyToLocal Command Example  
hdfs dfs -copyToLocal /abc/WC.txt /home/hduser/Desktop  
  
Similar to get command, only the difference is that in this the destination is restricted to a local file reference.

# 7. cat

Hadoop HDFS cat Command Usage  
cat  
Hadoop HDFS cat Command Example  
hdfs dfs -cat /abc/WC.txt  
  
  
This Hadoop fs shell command displays the contents of the filename on console or stdout.

# 8. mv

Hadoop HDFS mv Command Usage  
mv  
Hadoop HDFS mv Command Example  
hadoop fs -mv /abc /FFF  
hadoop fs -ls /FFF  
  
This basic HDFS command moves the file or directory indicated by the source to destination, within HDFS.

# 9. cp

Hadoop HDFS cp Command Usage  
cp  
Hadoop HDFS cp Command Example  
hadoop fs -cp /CSE/ /LLL  
hadoop fs -ls /LLL  
The cp command copies a file from one directory to another directory within the HDFS.

|  |
| --- |
|  |

// start hadoop (must be in hduser)

$ start-all.sh

// creating a directory inside hadoop -mkdir

$ hdfs dfs -mkdir /bda\_hadoop

// listing all content inside hadoop - ls

$ hadoop fs -ls /

// copyig files from deskop using put command - put

$ hdfs dfs -put /home/hduser/Desktop/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 using copyFromLocal cmd.

$ hdfs dfs -copyFromLocal /home/hduser/Desktop/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/hduser/Downloads/downloaded\_file.txt

$ hdfs dfs -getmerge /bda\_hadoop/file.txt /bda\_hadoop/file\_cp\_local.txt /home/hduser/Downloads/downloaded\_file.txt

$ hadoop fs -getfacl /bda\_hadoop/

# file: /bda\_hadoop

# owner: hduser

# group: supergroup

user::rwx

group::r-x

other::r-x

// copyToLocal

$ hdfs dfs -copyToLocal /bda\_hadoop/file.txt /home/hduser/Desktop

// mv command

$ hadoop fs -mv /bda\_hadoop /abc

$ hadoop fs -ls /abc

Found 1 items

drwxr-xr-x - hduser supergroup 0 2022-06-06 11:52 /abc/bda\_hadoop

// copy

$ hadoop fs -cp /hello/ /hadoop\_lab

**6. 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.**

**Dataset: https://github.com/tomwhite/hadoop-book/tree/master/input/ncdc/all**

**Driver code:**

package averagetemp\_amit;

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);

}

}

**Mapper:**

package averagetemp\_amit;

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));

}

}

**Reducer:**

package averagetemp\_amit;

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));

}

}

## **7. Create a Map Reduce program to sort the content in an alphabetic order listing only top 10 maximum occurrences of words.**

**Driver:**

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);

}

}

}

}

**Mapper:**

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);

}

}

}

**Combiner:**

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));

}

}

**Reducer:**

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));

}

}

}

**MiscUtils.java**

package utils;

import java.util.\*;

public class MiscUtils {

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());

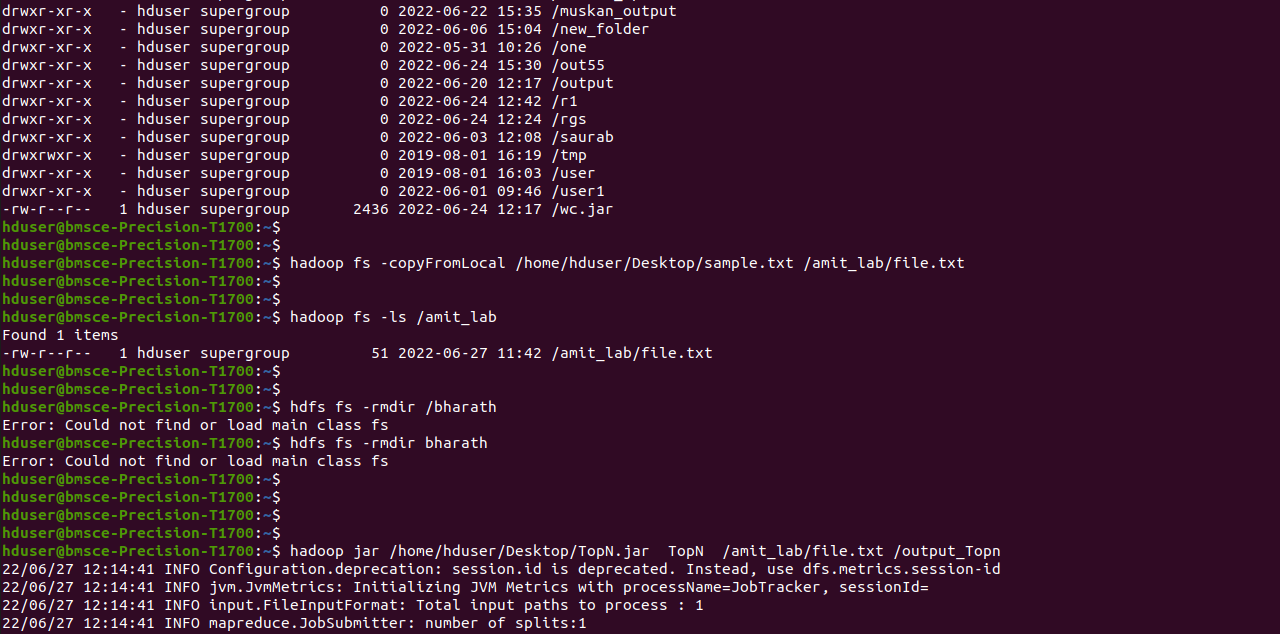
}

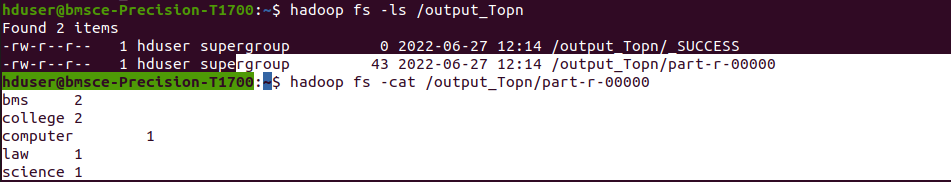
return sortedMap;

}

}

**Output:**





## **8. Create a Map Reduce program to demonstrating join operation.**

**DeptEmpStrength.txt**

Dept\_ID Total\_Employee

A11 50

B12 100

C13 250

**DeptName.txt**

Dept\_ID Dept\_Name

A11 Finance

B12 HR

C13 Manufacturing

**Driver:**

package MapReduceJoin;

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]);

MultipleInputs.addInputPath(conf, AInputPath, TextInputFormat.class, DeptNameMapper.class);

MultipleInputs.addInputPath(conf, BInputPath, TextInputFormat.class, DeptEmpStrengthMapper.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);

}

}

**Mapper:**

DeptEmpStrengthMapper.java

package MapReduceJoin;

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 DeptEmpStrengthMapper 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]));

}

}

DeptNameMapper.java

package MapReduceJoin;

import java.io.IOException;

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

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

public class DeptNameMapper 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], "0"), new Text(SingleNodeData[1]));

}

}

**Reducer:**

package MapReduceJoin;

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);

}

}

}

Jar link:

<https://github.com/amitkumar70512/BDA_LAB/blob/main/Lab8/MapReduceJoin/MapReduceJoin.jar>

## **9. Program to print word count on scala shell and print “hello world” on scala IDE.**

**scala program to print "Hello World".**

object printNumbers {

def main(args: Array[String]) {

println("Hello World!")

}

}

**output**

Hello World!

**Word count using scala**

### we find and display the number of occurrences of each word.

$ hdfs dfs -mkdir /spark

$ hdfs dfs -put /home/amit/sparkdata.txt /spark

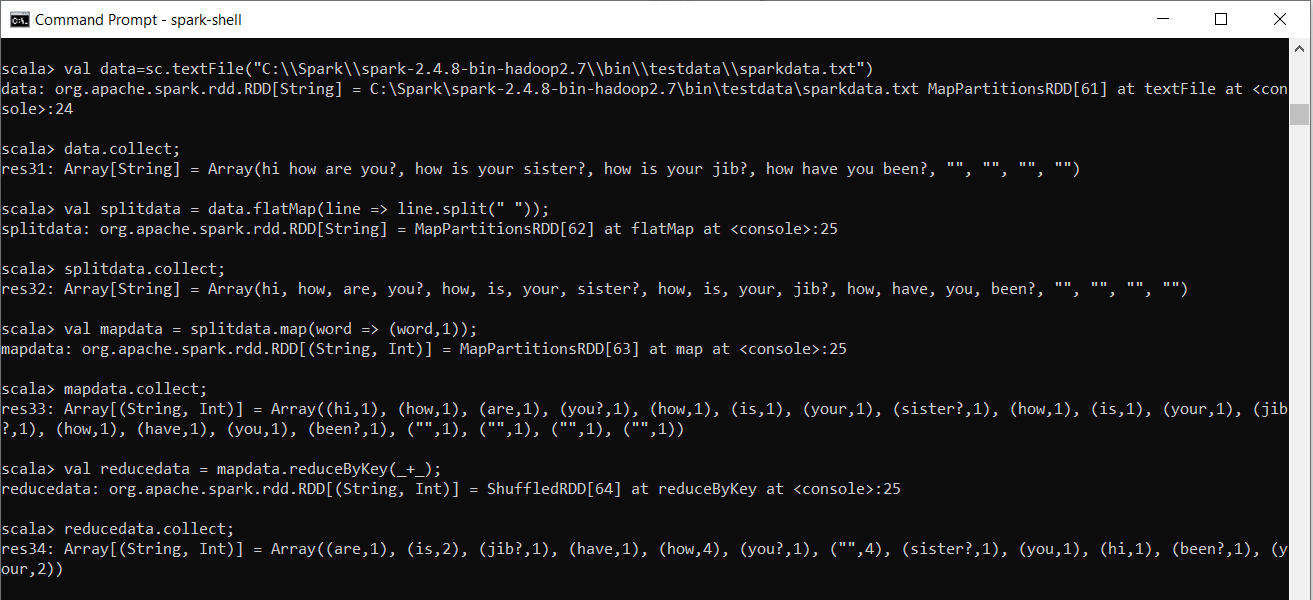
scala> val data=sc.textFile("sparkdata.txt") .

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

scala> splitdata.collect;

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

scala> val reducedata = mapdata.reduceByKey(+);



## **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.**

import org.apache.spark.SparkConf

import org.apache.spark.SparkContext

import org.apache.spark.rdd.RDD.rddToPairRDDFunctions

object WordCount {

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

*//Start the Spark context*

val conf = new SparkConf().setAppName("WordCount").setMaster("local")

val sc = new SparkContext(conf)

*//Read some example file to a test RDD*

val test = sc.textFile("input.txt")

test.flatMap { line => line.split(" ") *//split the line in word by word.*

}.map {

word => (word, 1) *//Return a key/value tuple, with the word as key and 1 as value*

}.reduceByKey(\_ + \_).saveAsTextFile("output.txt") *//Save to a text file*

sc.stop *//Stop the Spark context*

}

}

