**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

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

****

**LAB REPORT**

**on**

**BIG DATA ANALYTICS**

**(20CS6PEBDA)**

***Submitted by***

**N.AKHILESH KUMAR DUTT (1BM19CS092)**

***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 **N.AKHILESH KUMAR DUTT(1BM19CS092),** 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.

Dr.Pallavi G B               **Dr. Jyothi S Nayak**

Assistant Professor Professor and Head

Department of CSE Department of CSE

BMSCE, Bengaluru BMSCE, Bengaluru

`

**Index Sheet**

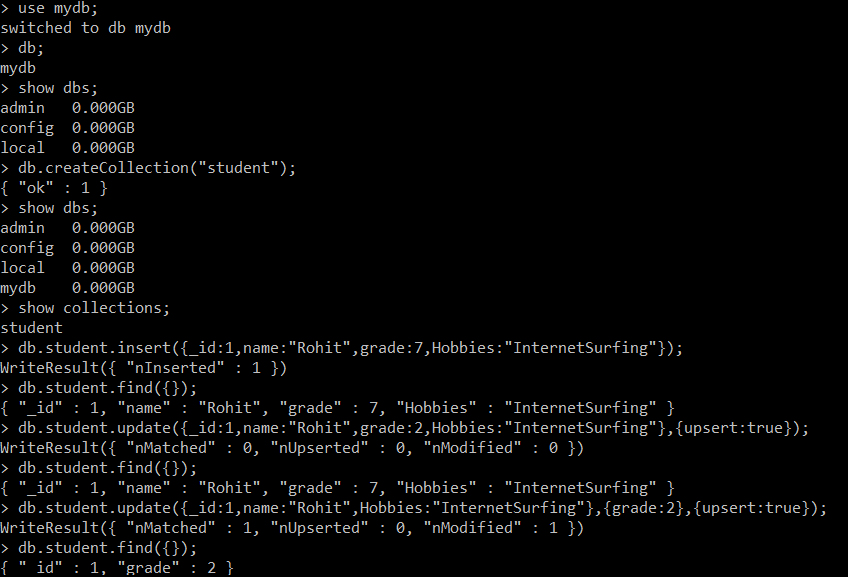
|  |  |  |
| --- | --- | --- |
| **Sl. No.** | **Experiment Title** | **Page No.** |
| **1** | **MongoDB CRUD Demonstration** | **4** |
| **2** | **Cassandra Employee Database** | **6** |
| **3** | **Cassandra Library Database** | **8** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

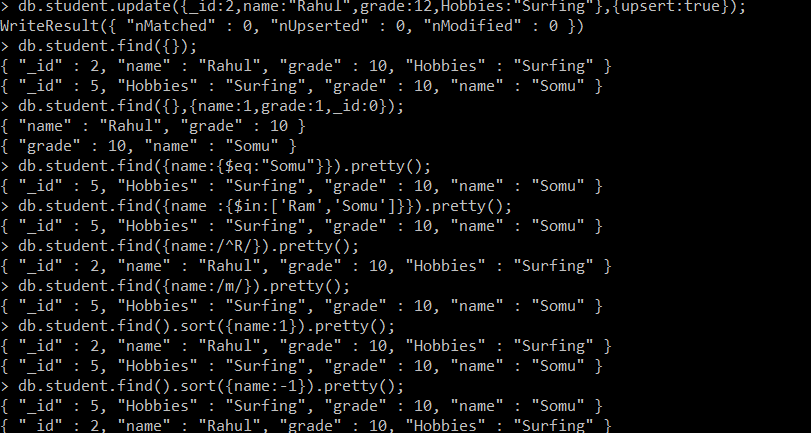
**Course Outcome**

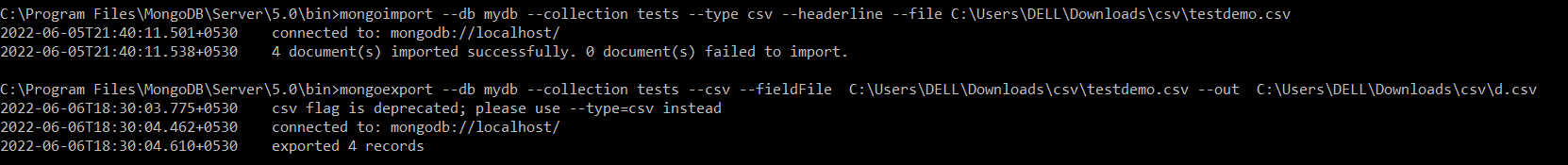
|  |  |
| --- | --- |
| CO1 | Apply the concept of NoSQL, Hadoop or Spark for a given task |
| CO2 | Analyze the Big Data and obtain insight using data analytics mechanisms. |
| CO3 | Design and implement Big data applications by applying NoSQL, Hadoop or Spark |

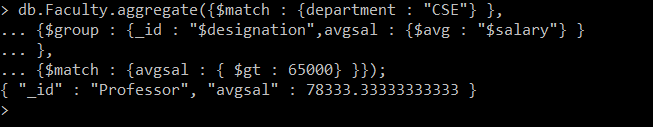
1. **MongoDB- CRUD Demonstration**

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

****

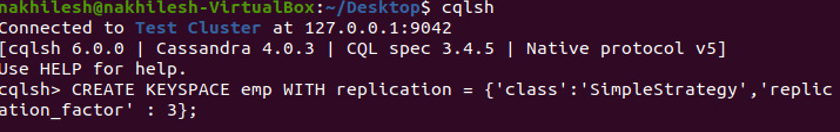
****

****

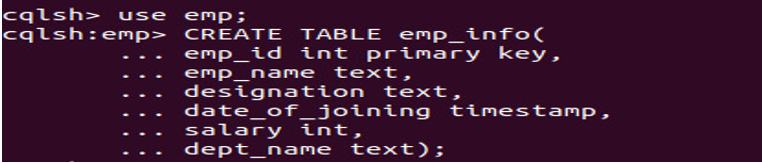
****

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

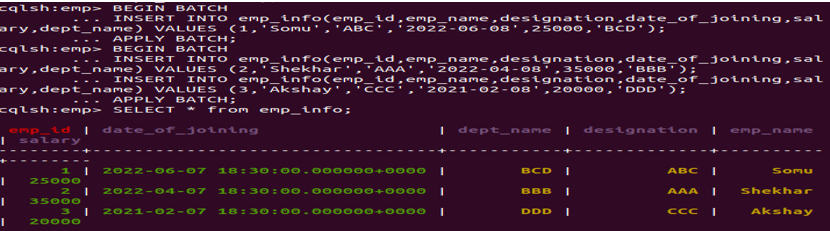
1.Create a key space by name Employee

****

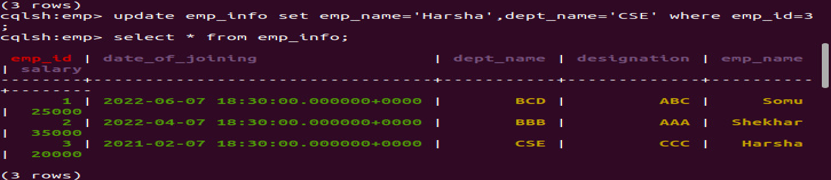
2.Create a column family by name Employee-Info with attributes Emp\_Id Primary Key, Emp\_Name, Designation, Date\_of\_Joining, Salary, Dept\_Name

****

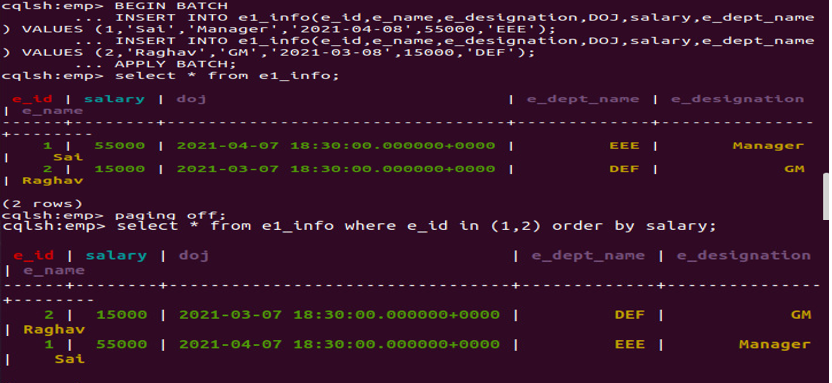
3.Insert the values into the table in batch

****

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

****

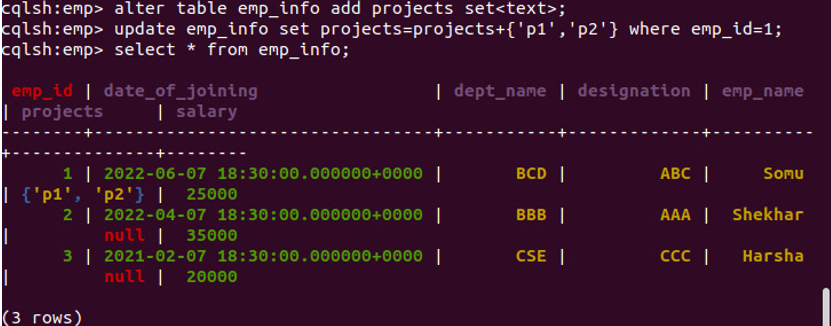
5. Sort the details of Employee records based on salary

****

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

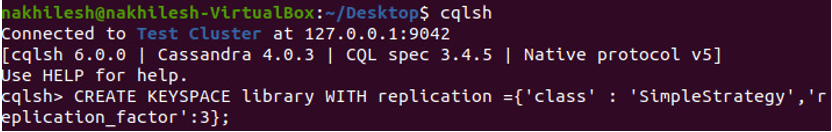
****

7. Update the altered table to add project names

****

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

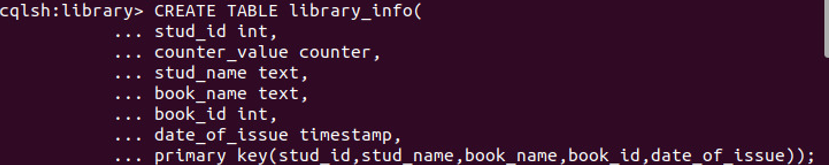
1. Create a key space by name Library

****

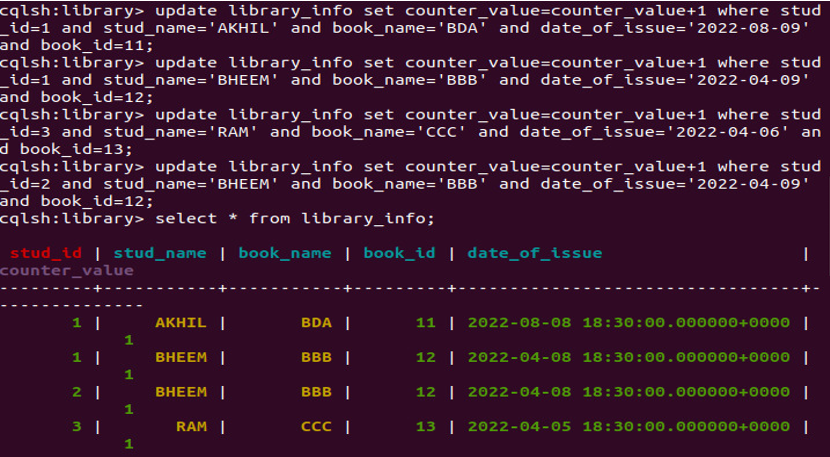
2. Create a column family by name Library-Info with attributes Stud\_Id Primary Key,

Counter\_value of type Counter,

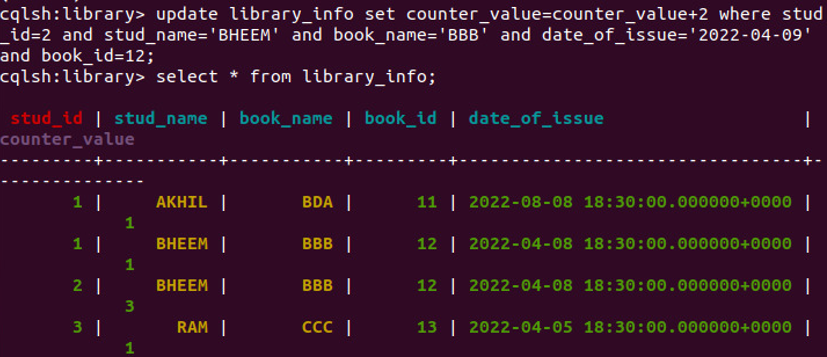
Stud\_Name, Book-Name, Book-Id, Date\_of\_issue

****

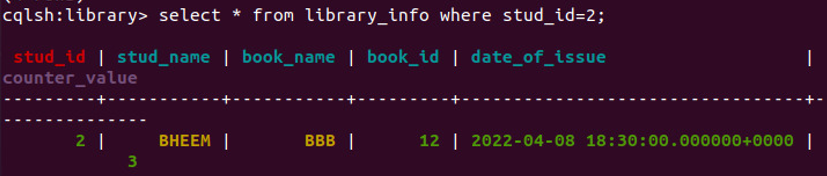
3. Insert the values into the table in batch

****

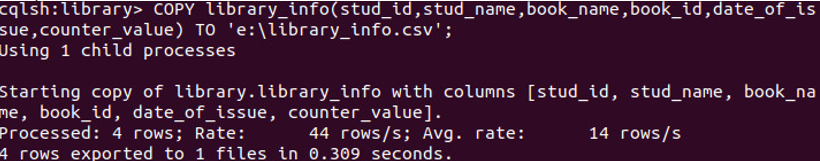
4.Display the details of the table created and increase the value of the counter

****

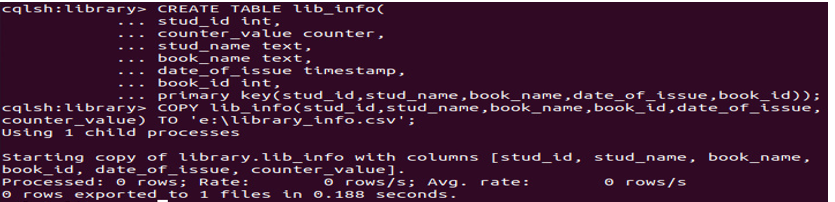
5. Write a query to show that a student with id 112 has taken a book “BDA” 2 times.

****

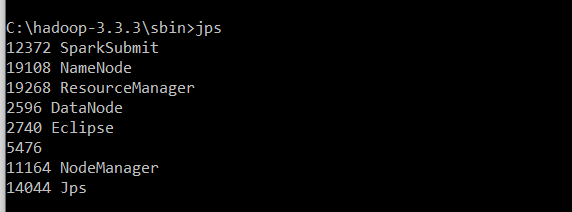
6. Export the created column to a csv file

****

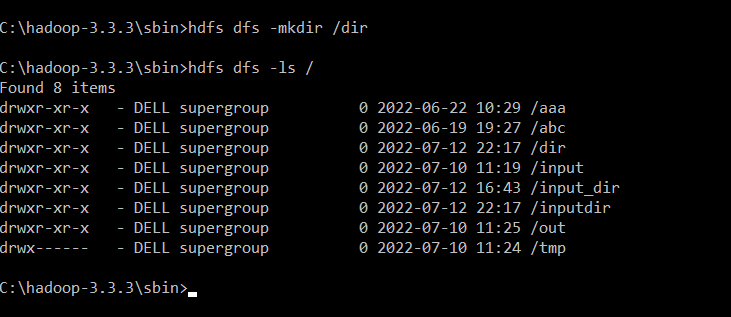
7. Import a given csv dataset from local file system into Cassandra column family

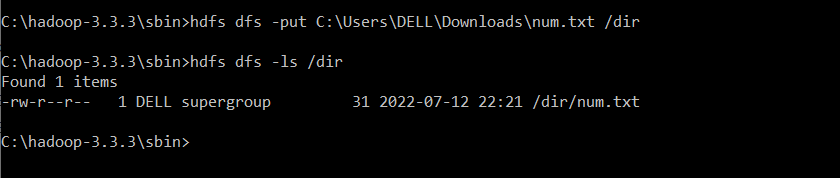
****

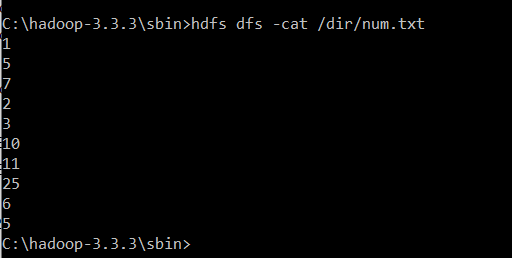
**4.Hadoop Installation**

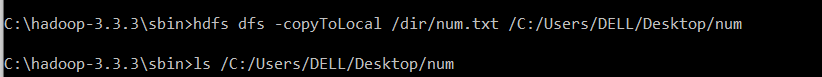
****

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

****

****

****

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

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

**Average Driver Class**

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);  
 }  
}

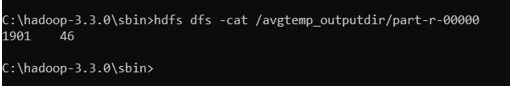
job.setOutputKeyClass(Text.class);  
 job.setOutputValueClass(IntWritable.class);  
 System.exit(job.waitForCompletion(true) ? 0 : 1);  
 }

**Average Mapper Class**

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));   
 }  
}

**Average Reducer class**

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));  
 }  
}

****

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

**MeanMax driver class**

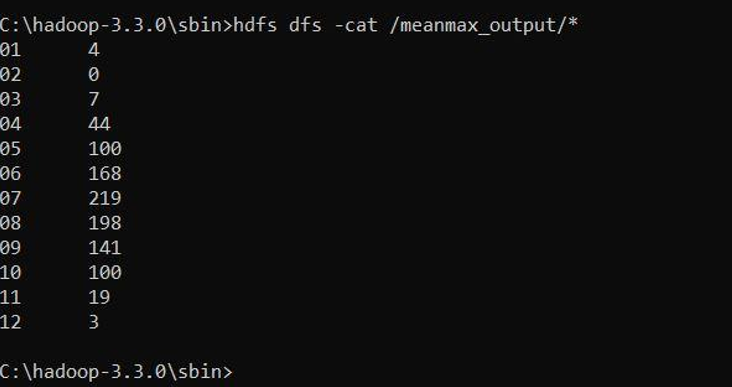
package meanmax;  
  
import org.apache.hadoop.fs.Path;  
import org.apache.hadoop.io.IntWritable;  
import org.apache.hadoop.io.Text;  
import org.apache.hadoop.mapreduce.Job;  
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;  
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
  
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);  
 }  
}

**MeanMax Mapper class**

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));   
 }  
}

**MeanMax Reducer Class**

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));  
 }  
}

****

**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.**

**TopN Driver Class**

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);  
 }   
 }  
 }  
}

**TopN Combiner Class**

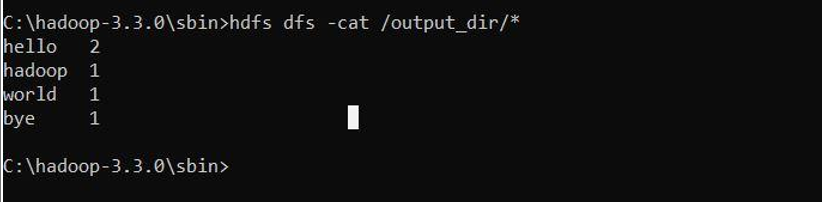
package samples.topn;  
  
import java.io.IOException;  
import org.apache.hadoop.io.IntWritable;  
import org.apache.hadoop.io.Text;  
import org.apache.hadoop.mapreduce.Reducer;  
  
public class TopNCombiner extends Reducer<Text, IntWritable, Text, IntWritable> {  
 public void reduce(Text key, Iterable<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));  
 }  
}

**TopN Mapper Class**

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 vo```\\id 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);  
 }   
 }  
}

**TopN Reducer Class**

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));  
 }   
 }  
}

****

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

// 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]);

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

Posts.class);

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

User.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) {

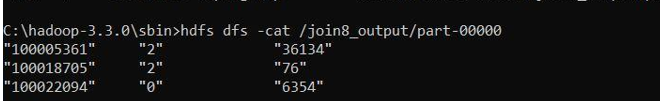
return ((TextPair) a).first.compareTo(((TextPair) b).first);

}

return super.compare(a, b);

}

} }

****

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

scala> println("Hello World!");

Hello World!

val data=sc.textFile("C:\Users\DELL\Downloads\new.txt ")

data.collect;

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

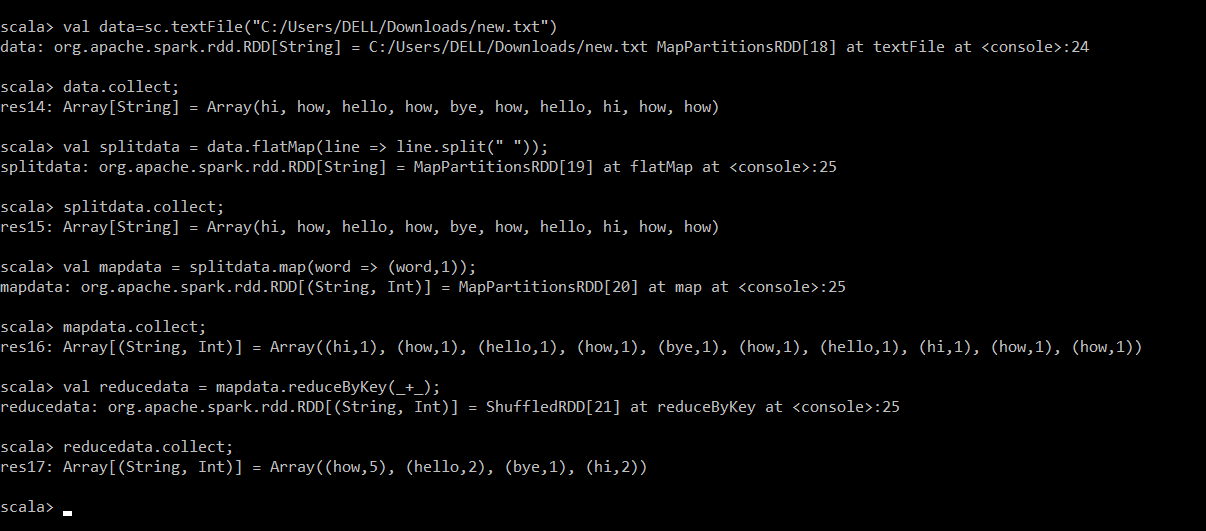
splitdata.collect;

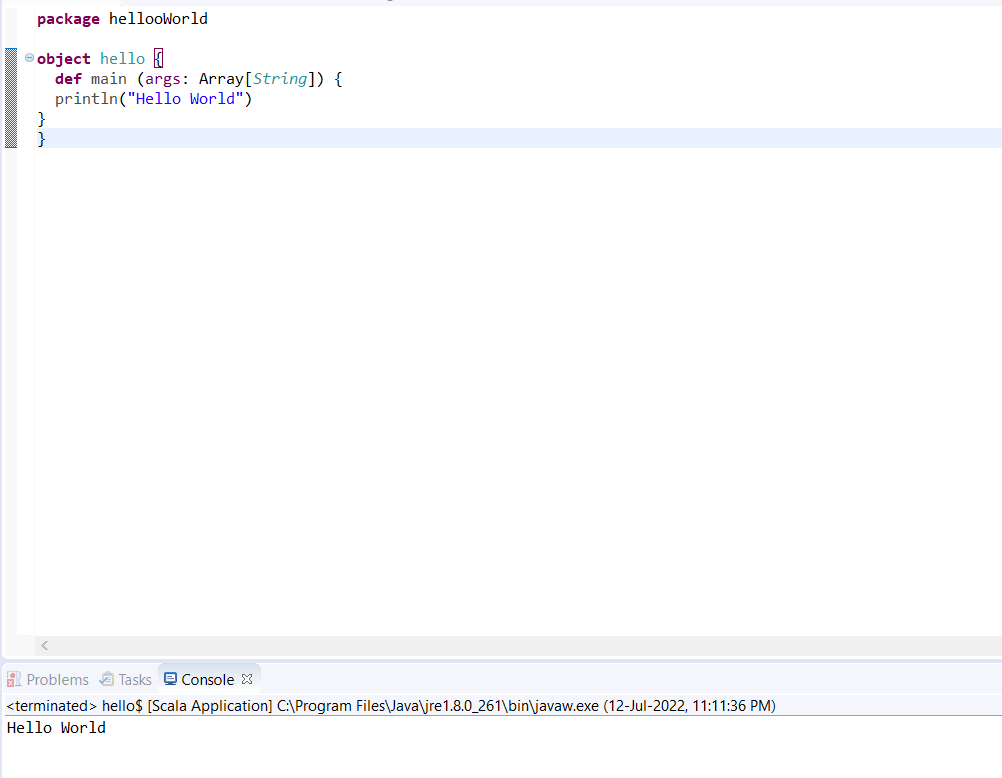
val mapdata = splitdata.map(word =**>** (word,1));

mapdata.collect;

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

reducedata.collect;



****

**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**

val textFile = sc.textFile("C:\Users\DELL\Downloads\new.txt ")

val counts = textFile.flatMap(line => line.split(" ")).map(word => (word, 1)).reduceByKey(\_ + \_)

import scala.collection.immutable.ListMap

val sorted=ListMap(counts.collect.sortWith(\_.\_2 > \_.\_2)

println(sorted)

for((k,v)<-sorted)

{

  if(v>4)

    {

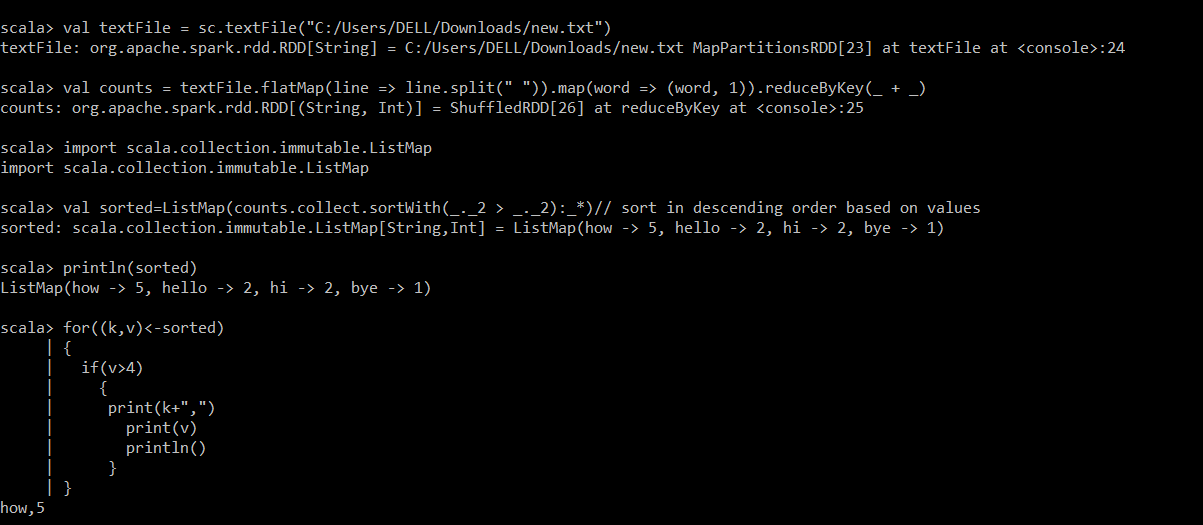
     print(k+",")

       print(v)

       println()

     }

}

****