**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

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

****

**LAB REPORT**

**on**

**Big Data Analytics (23CS6PCBDA)**

***Submitted by***

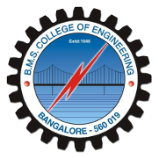
**MAHIKA D (1BM22CS142)**

***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 July-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” carried out by **MAHIKA D (1BM22CS142),** who is a 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 2025. The Lab report has been approved as it satisfies the academic requirements in respect of a **Big Data Analytics - (23CS6PEBDA)** work prescribed for the said degree.

**Prof. Leelavathi B**            **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 Operations | 1-4 |
| 2. | Neo4j | 4-5 |
| 3. | Cassandra Employee | 6-7 |
| 4. | Cassandra Library | 8-9 |
| 5. | Hadoop Installation | 10 |
| 6. | Hadoop Commands | 11-12 |
| 7. | Hadoop Word Count | 13-16 |
| 8. | Map Reduce Programs | 17-21 |
| 9. | Spark RDD and FlatMap | 22-25 |

**Course Outcome**

|  |  |
| --- | --- |
| CO1 | Apply the concepts of NoSQL, Hadoop, Spark for a given task |
| CO2 | Analyze data analytic techniques for a given problem |
| CO3 | Conduct experiments using data analytics mechanisms for a given problem |

**LAB 1**

**MongoDB- CRUD Demonstration**

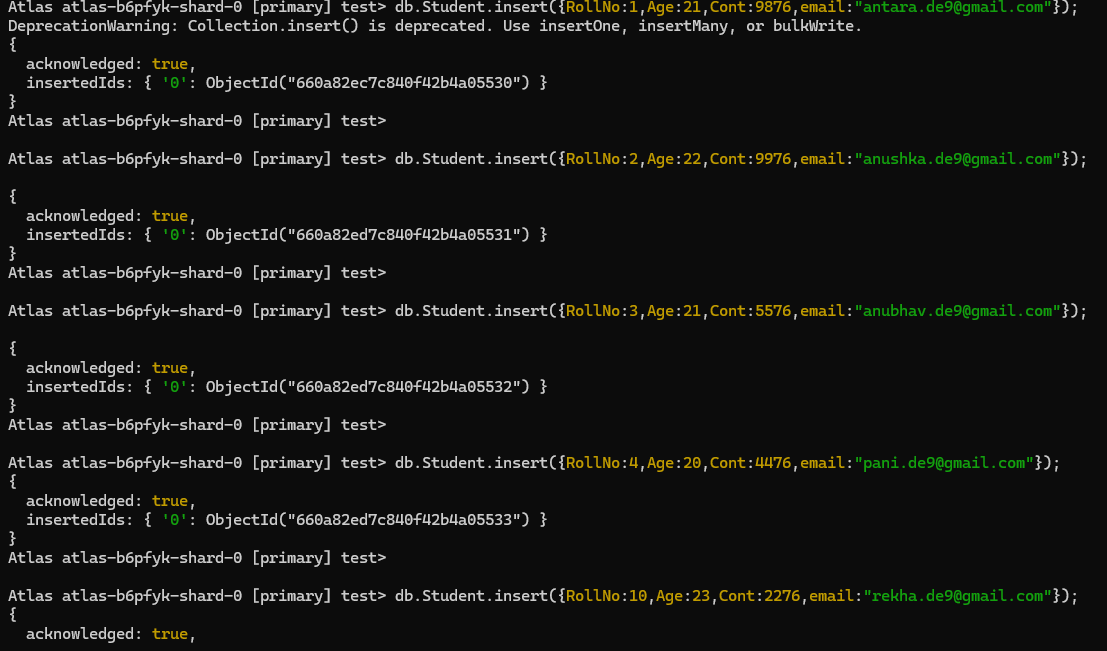
SETUP:



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

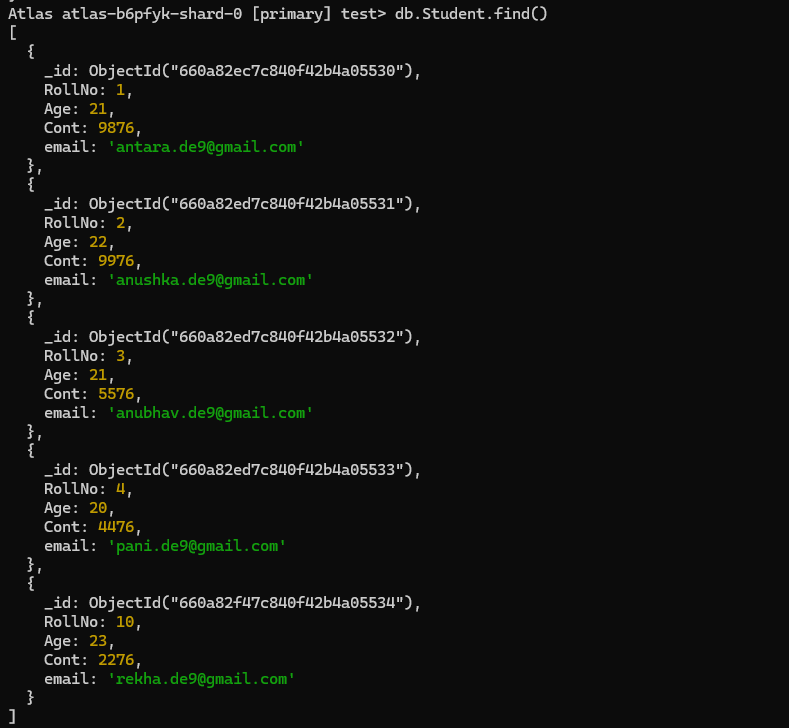


1. Insert appropriate values(at least 5)

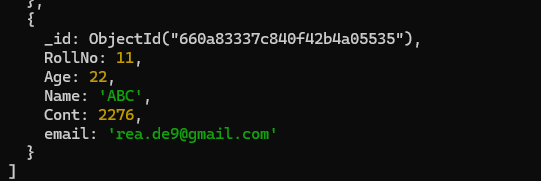




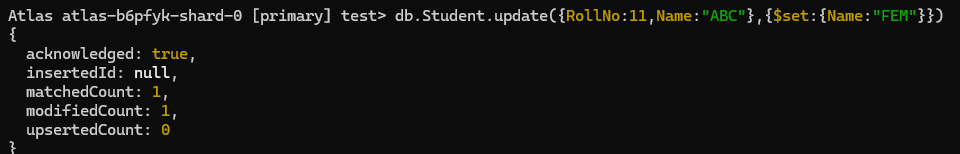
1. View the data



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



1. Replace the student name from “ABC” to “FEM” of rollno 11.



1. Drop the table



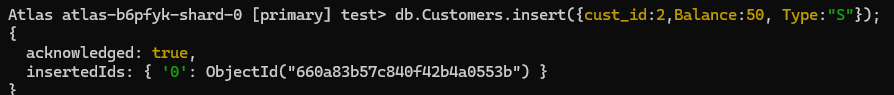
1. Create a collection by name Customers with the following attributes.

Cust\_id, Acc\_Bal, Acc\_Type

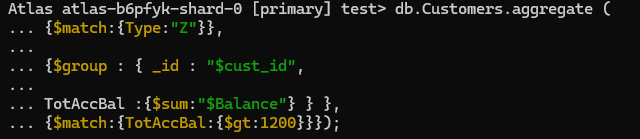


1. Insert at least 5 values into the table

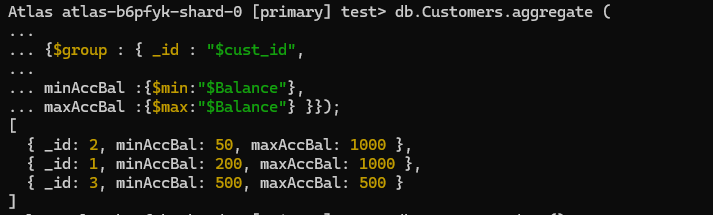




1. Write a query to display those records whose total account balance is greater than 1200 of account type ‘Z’ for each customer\_id.



1. Determine Minimum and Maximum account balance for each customer\_id.



1. Drop the table

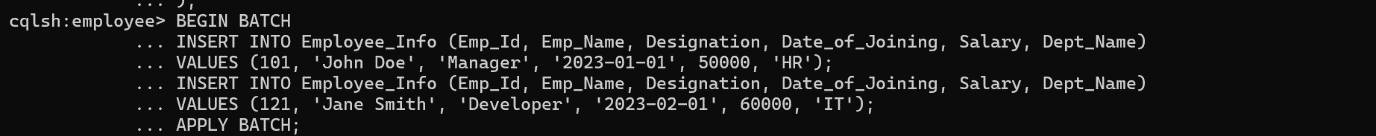


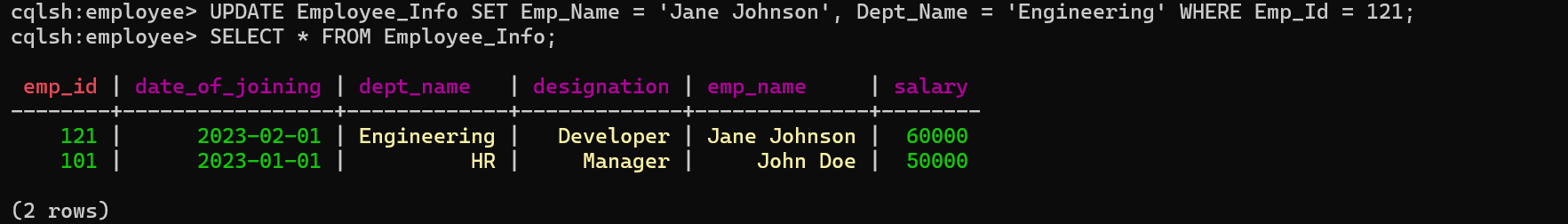
**LAB 2**

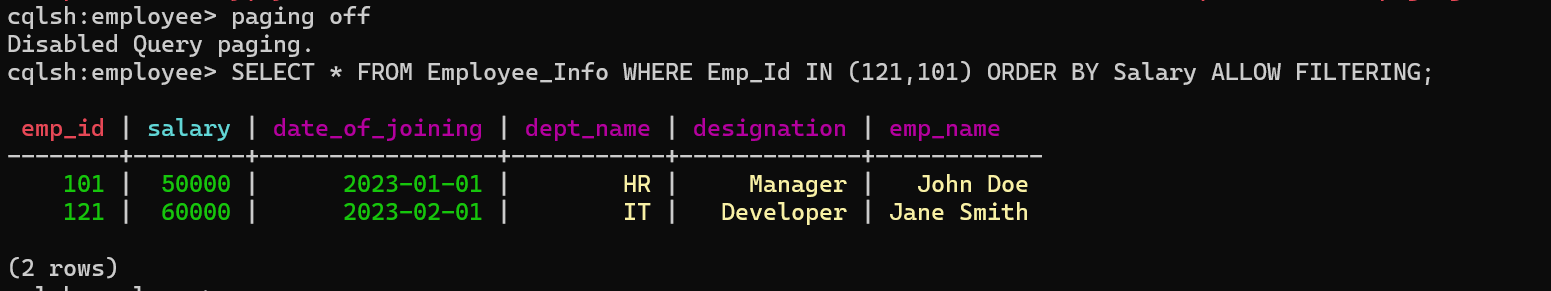
**Perform the following DB operations using Cassandra**

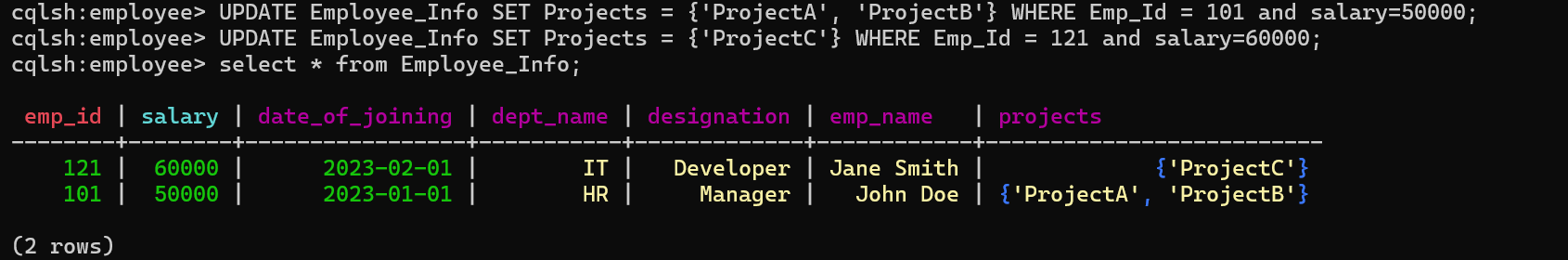
1. Create a keyspace by name Employee 
2. Create a column family by name Employee-Info with attributes Emp\_Id Primary Key, Emp\_Name, Designation, Date\_of\_Joining, Salary, Dept\_Name 

3. Insert the values into the table in batch

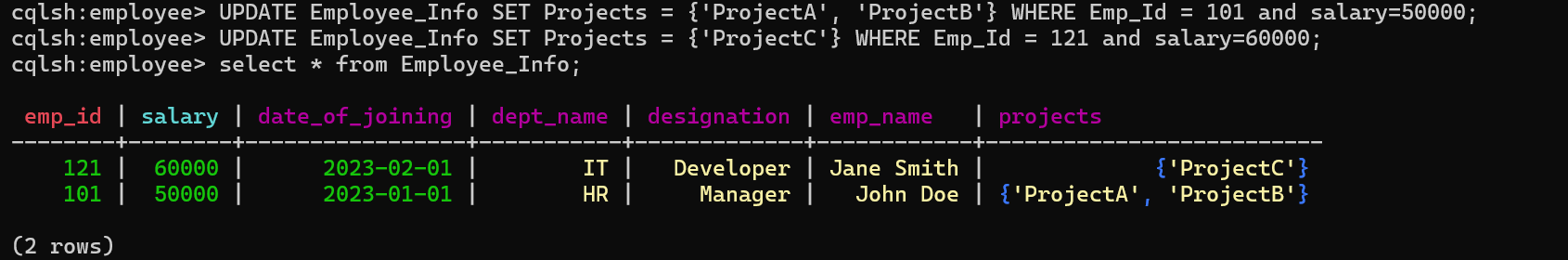


4. Update Employee name and Department of Emp-Id 121

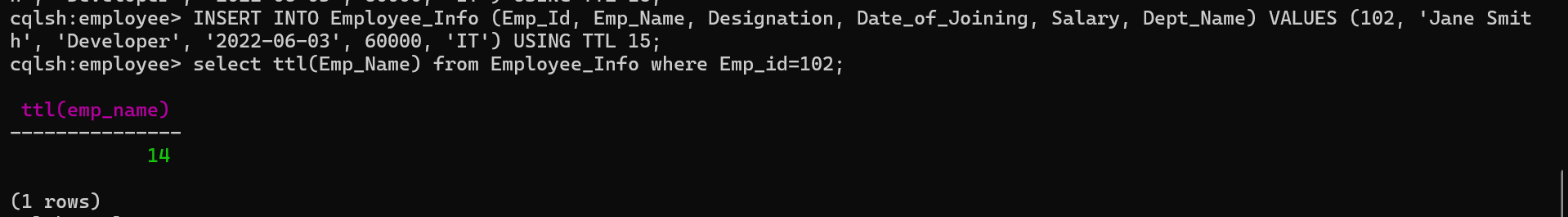
5. Sort the details of Employee records based on salary 

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.



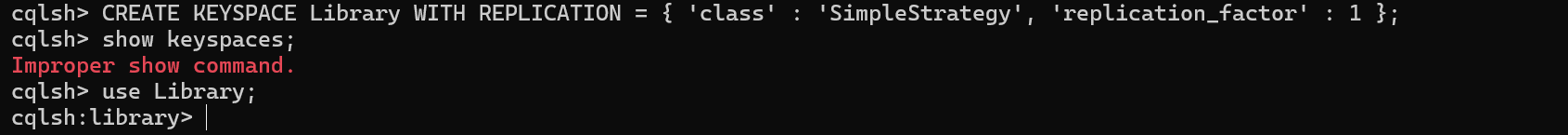
8. Create a TTL of 15 seconds to display the values of Employees.



**LAB 3**

**Perform the following DB operations using Cassandra**

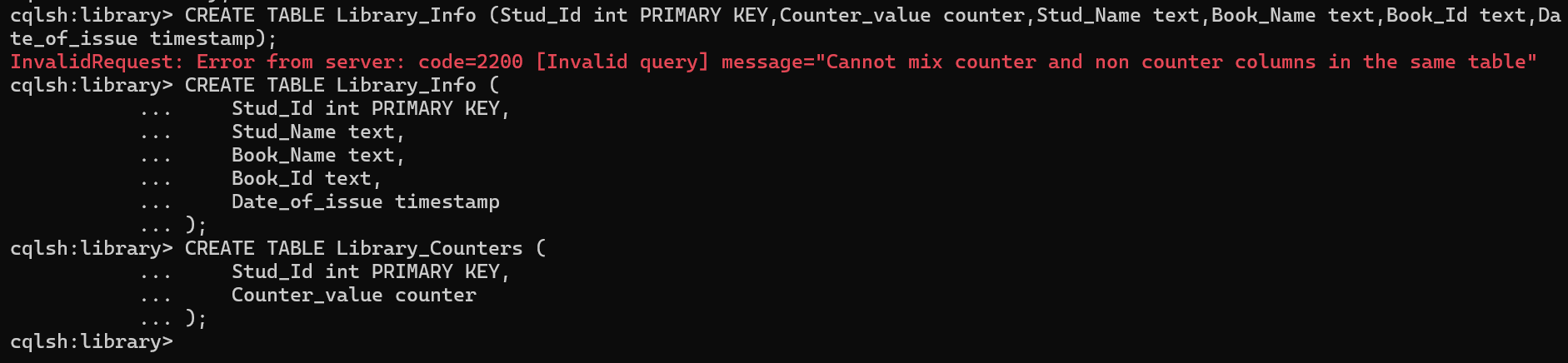
1. Create a keyspace by name Library



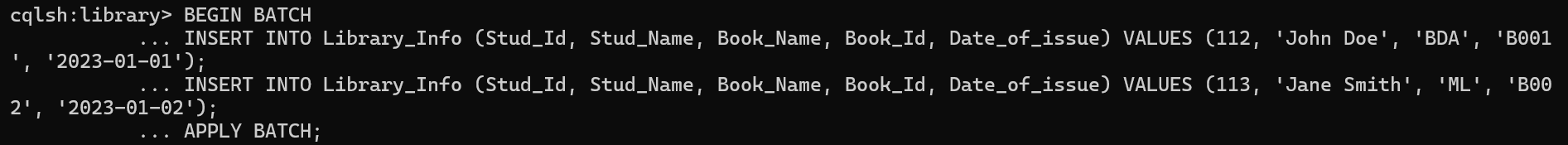
2. Create a column family by name Library-Info with attributes

Stud\_Id Primary Key, Counter\_value of type Counter,

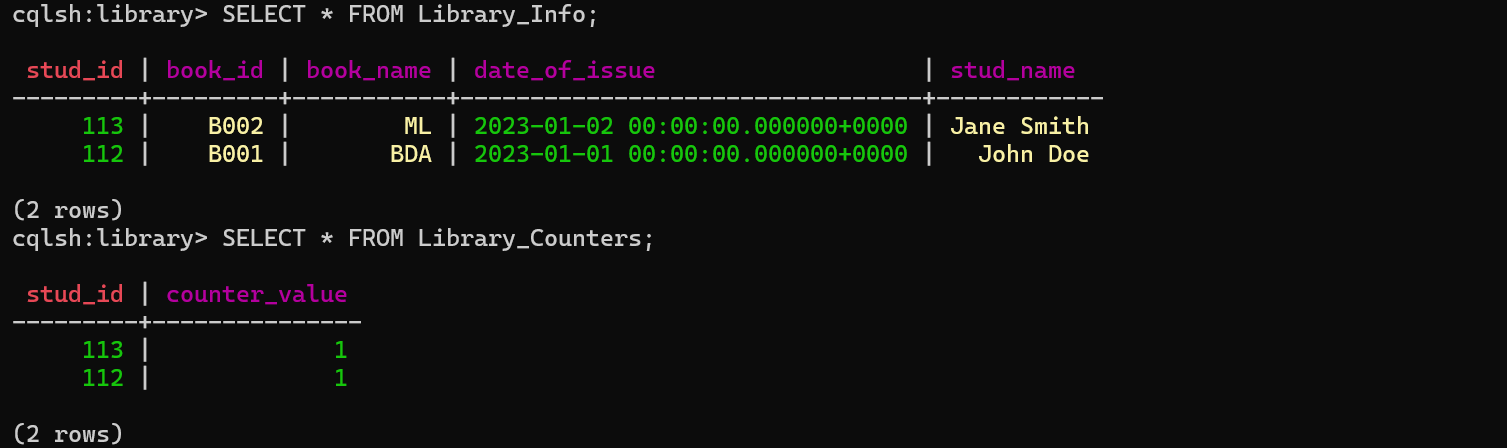
Stud\_Name, Book-Name, Book-Id, Date\_of\_issue



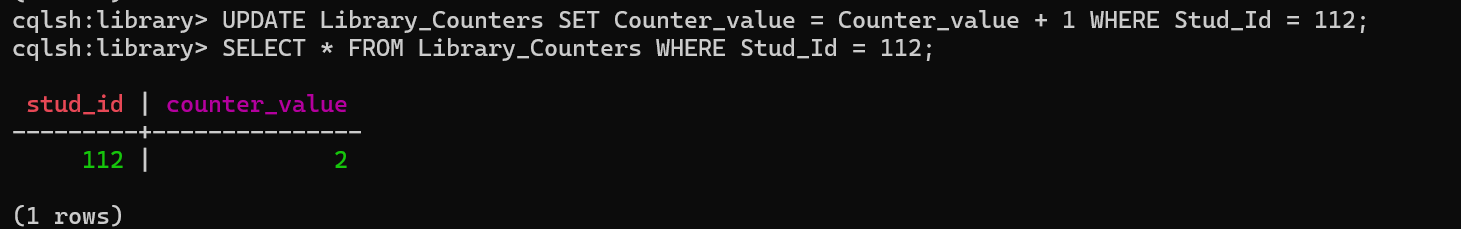
3. Insert the values into the table in batch



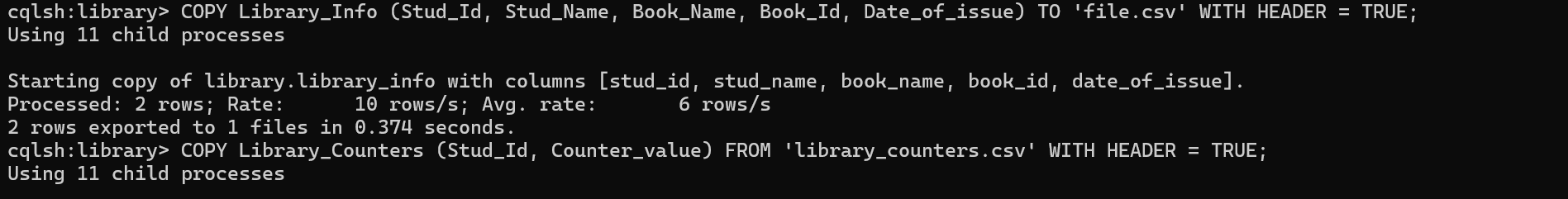
4. Display the details of the table created and increase the value of the counter



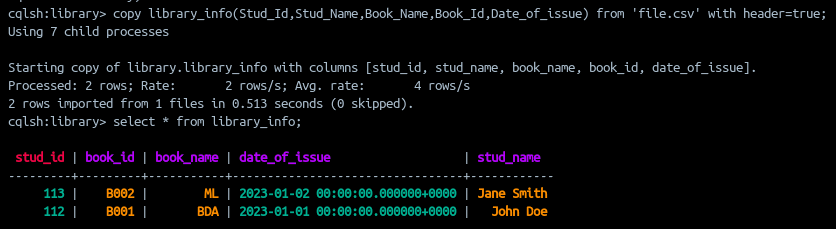
5. Write a query to show that a student with id 112 has taken a book “BDA” 2 times.



6. Export the created column to a csv file

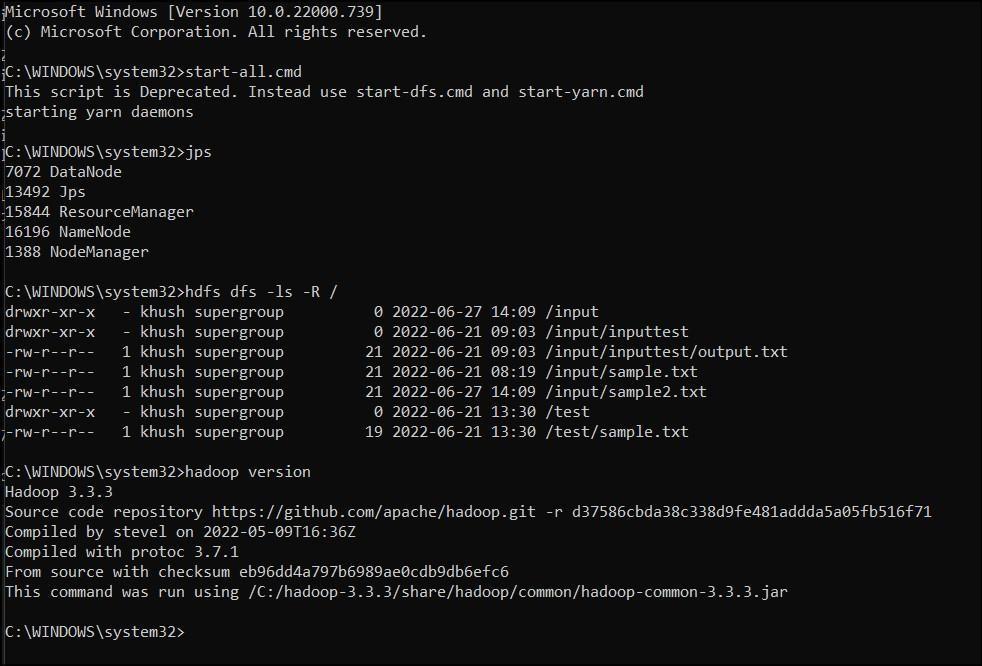


7. Import a given csv dataset from local file system into Cassandra column family



**LAB 4**

**Screenshot of Hadoop installed**



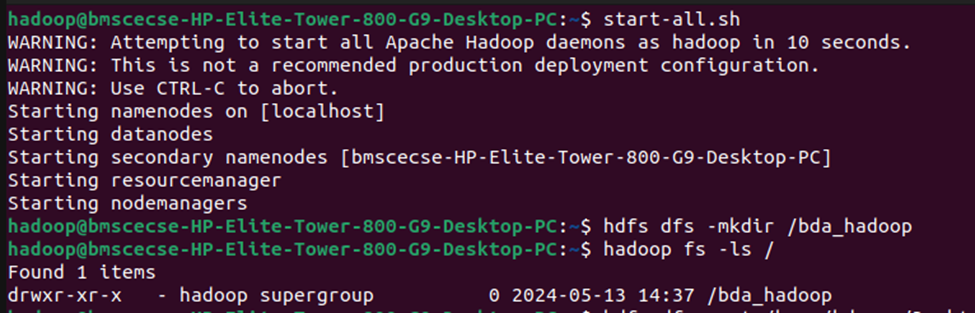
**LAB 5**

**Execution of HDFS Commands for interaction with Hadoop**

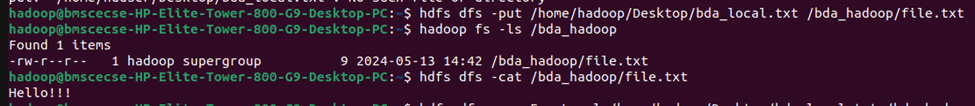
**Environment. (Minimum 10 commands to be executed)**

1. mkdir

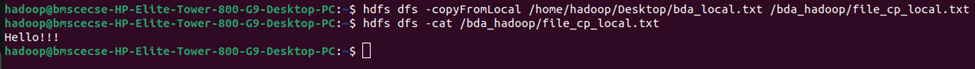
2.ls



3.put

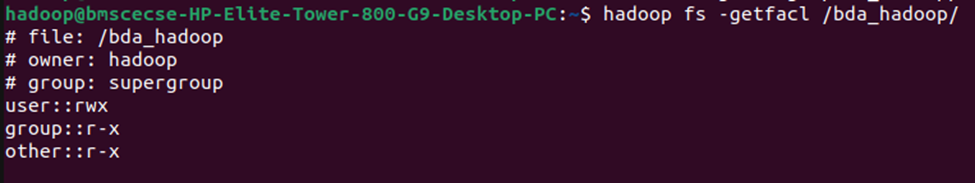


4. copyFromLocal

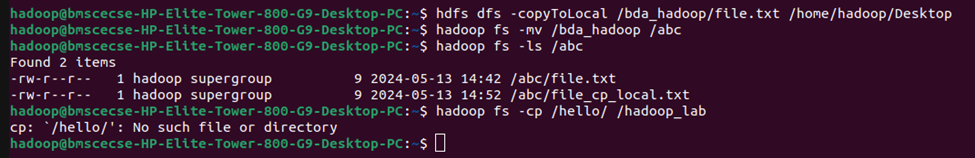


5. get

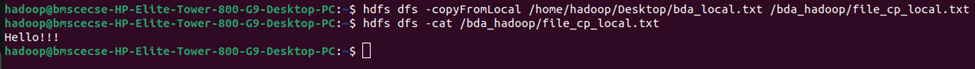




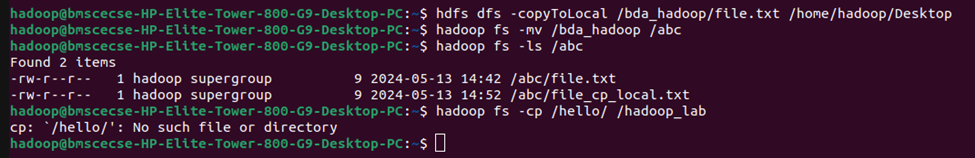
6. copyToLocal



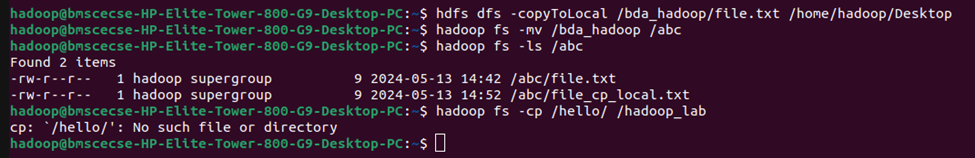
7. cat



8.mv



9.cp



**LAB 6**

**Implement WordCount Program on Hadoop framework**

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&lt;LongWritable,

Text, Text,

IntWritable&gt; {

// Map function

public void map(LongWritable key, Text value, OutputCollector&lt;Text,

IntWritable&gt; output, Reporter rep) throws IOException

{

String line = value.toString();

// Splitting the line on spaces

for (String word : line.split(&quot; &quot;))

{

if (word.length() &gt; 0)

{

output.collect(new Text(word), new IntWritable(1));

} } } }

Reducer Code: You have to copy paste this program into the WCReducer Java Class file

// Importing libraries

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&lt;Text,

IntWritable, Text, IntWritable&gt; {

// Reduce function

public void reduce(Text key, Iterator&lt;IntWritable&gt; value,

OutputCollector&lt;Text, IntWritable&gt; output,

Reporter rep) throws IOException

{

int count = 0;

// Counting the frequency of each words

while (value.hasNext())

{

IntWritable i = value.next();

count += i.get();

}

output.collect(key, new IntWritable(count));

} }

Driver Code: You have to copy paste this program into the WCDriver Java Class file.

// Importing libraries

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 &lt; 2)

{

System.out.println(&quot;Please give valid inputs&quot;);

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;

}

// Main Method

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

{

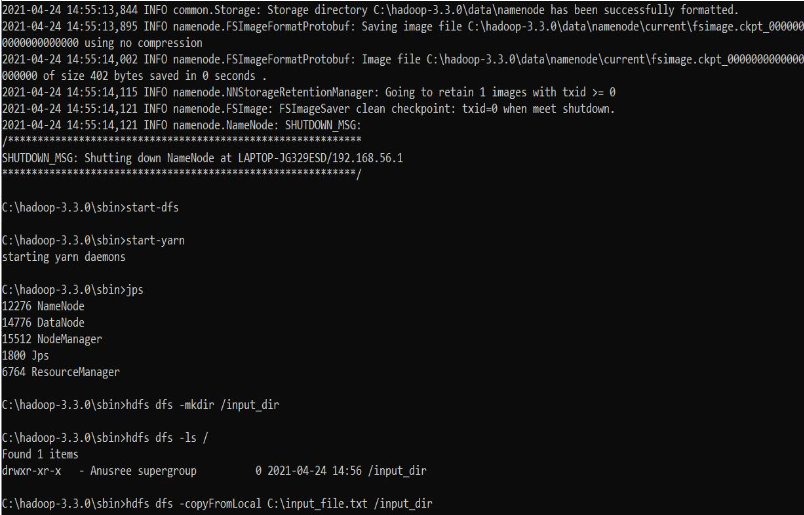
int exitCode = ToolRunner.run(new WCDriver(), args);

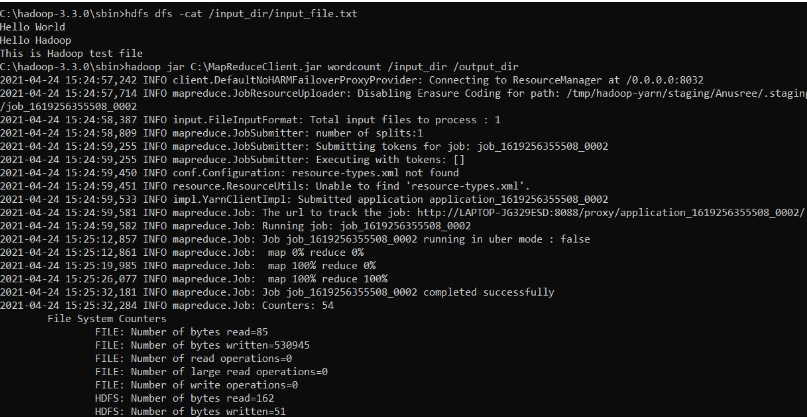
System.out.println(exitCode);

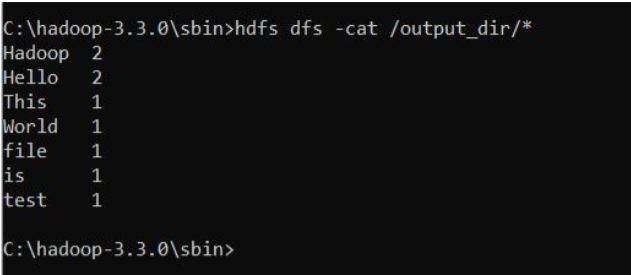
}

}

OUTPUT







**LAB 7**

**From the following link extract the weather data** [**https://github.com/tomwhite/hadoo**](https://github.com/tomwhite/hadoop-)**p-Book/tree/master/input/ncdc/all**

**Create a Map Reduce program to**

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

**AverageDriver**

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**

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&lt;LongWritable, Text, Text, IntWritable&gt; {

public static final int MISSING = 9999;

public void map(LongWritable key, Text value, Mapper&lt;LongWritable, Text, Text,

IntWritable&gt;.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 &amp;&amp; quality.matches(“[01459]”;))

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

}

}

**AverageReducer**

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&lt;Text, IntWritable, Text, IntWritable&gt; {

public void reduce(Text key, Iterable&lt;IntWritable&gt; values, Reducer&lt;Text, IntWritable,

Text, IntWritable&gt;.Context context) throws IOException, InterruptedException {

int max\_temp = 0;

int count = 0;

for (IntWritable value : values) {

max\_temp += value.get();

count++;

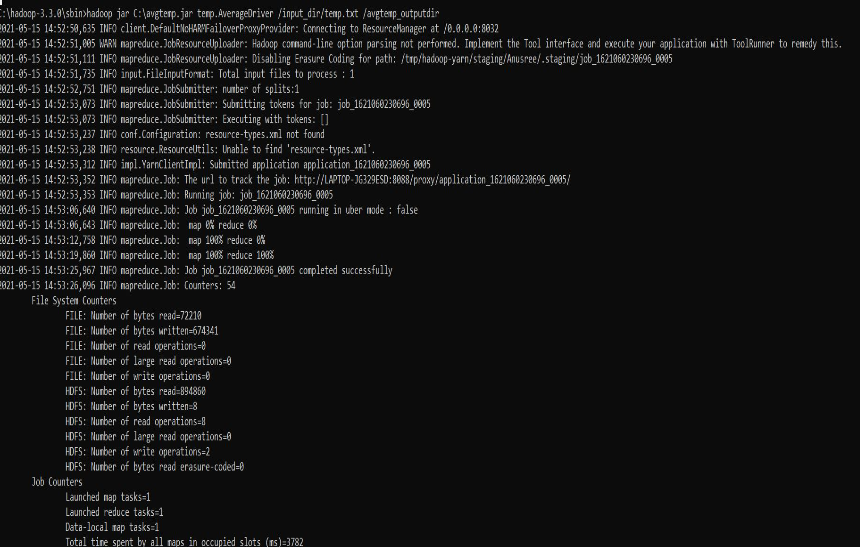
}

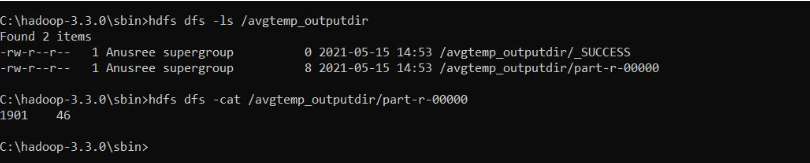
context.write(key, new IntWritable(max\_temp / count));

}

}

OUTPUT





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

**MeanMaxDriver.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);

}

}

**MeanMaxMapper.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&lt;LongWritable, Text, Text, IntWritable&gt; {

public static final int MISSING = 9999;

public void map(LongWritable key, Text value, Mapper&lt;LongWritable, Text, Text,

IntWritable&gt;.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 &amp;&amp; quality.matches(“[01459]"))

context.write(new Text(month), new IntWritable(temperature));

}

}

**MeanMaxReducer.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&lt;Text, IntWritable, Text, IntWritable&gt; {

public void reduce(Text key, Iterable&lt;IntWritable&gt; values, Reducer&lt;Text, IntWritable,

Text, IntWritable&gt;.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 &gt; max\_temp)

max\_temp = temp;

count++;

if (count == 3) {

total\_temp += max\_temp;

max\_temp = 0;

count = 0;

days++;

}

}

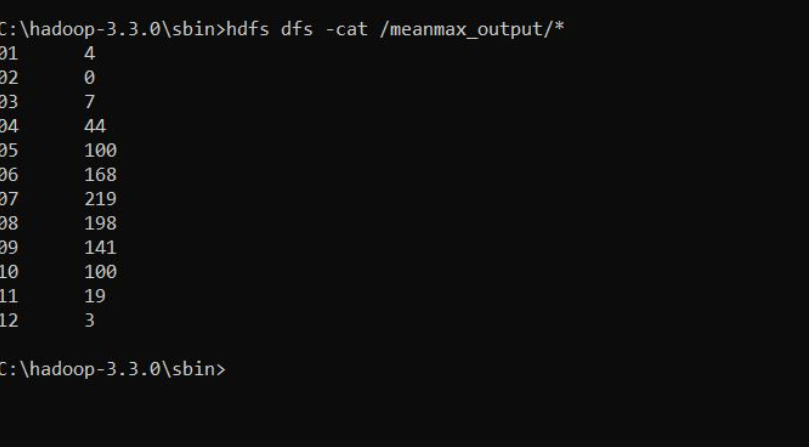
context.write(key, new IntWritable(total\_temp / days));

}

}

OUTPUT





**LAB 8**

**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**

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&lt;Object, Text, Text, IntWritable&gt; {

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

private Text word = new Text();

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

public void map(Object key, Text value, Mapper&lt;Object, Text, Text, IntWritable&gt;.Context

context) throws IOException, InterruptedException {

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

StringTokenizer itr = new StringTokenizer(cleanLine);

while (itr.hasMoreTokens()) {

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

context.write(this.word, one);

}

}

}

}

TopNCombiner.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&lt;Text, IntWritable, Text, IntWritable&gt; {

public void reduce(Text key, Iterable&lt;IntWritable&gt; values, Reducer&lt;Text, IntWritable,

Text, IntWritable&gt;.Context context) throws IOException, InterruptedException {

int sum = 0;

for (IntWritable val : values)

sum += val.get();

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

}

}

TopNMapper.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&lt;Object, Text, Text, IntWritable&gt; {

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

private Text word = new Text();

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

public void map(Object key, Text value, Mapper&lt;Object, Text, Text, IntWritable&gt;.Context

context) throws IOException, InterruptedException {

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

StringTokenizer itr = new StringTokenizer(cleanLine);

while (itr.hasMoreTokens()) {

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

context.write(this.word, one);

}

}

}

TopNReducer.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&lt;Text, IntWritable, Text, IntWritable&gt; {

private Map&lt;Text, IntWritable&gt; countMap = new HashMap&lt;&gt;();

public void reduce(Text key, Iterable&lt;IntWritable&gt; values, Reducer&lt;Text, IntWritable,

Text, IntWritable&gt;.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&lt;Text, IntWritable, Text, IntWritable&gt;.Context context)

throws IOException, InterruptedException {

Map&lt;Text, IntWritable&gt; sortedMap = MiscUtils.sortByValues(this.countMap);

int counter = 0;

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

if (counter++ == 20)

break;

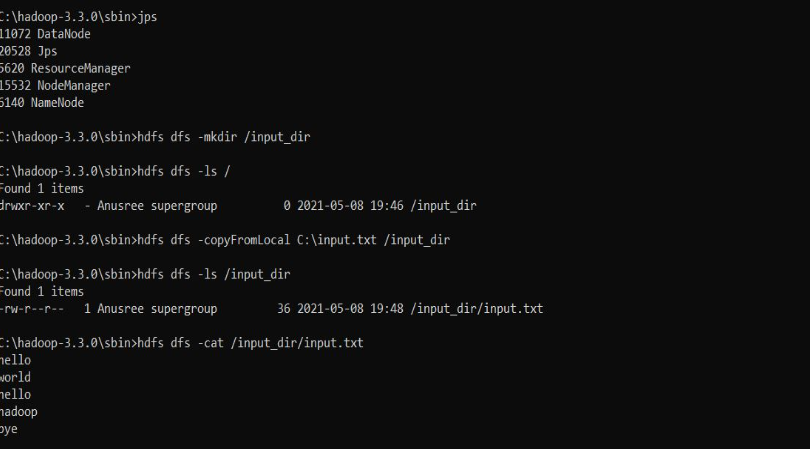
context.write(key, sortedMap.get(key));

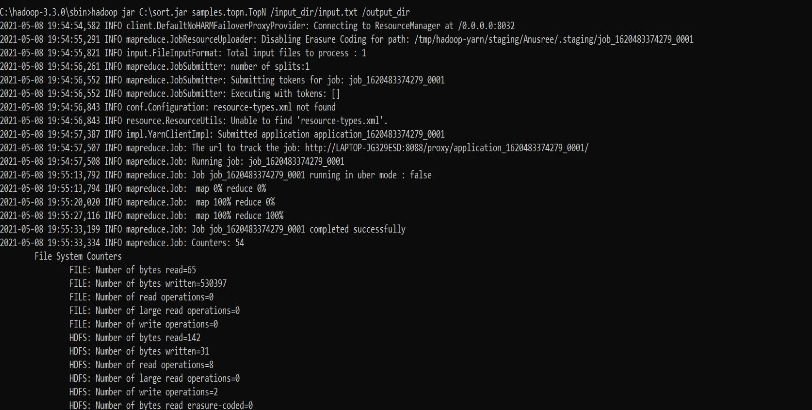
}

}

}

OUTPUT



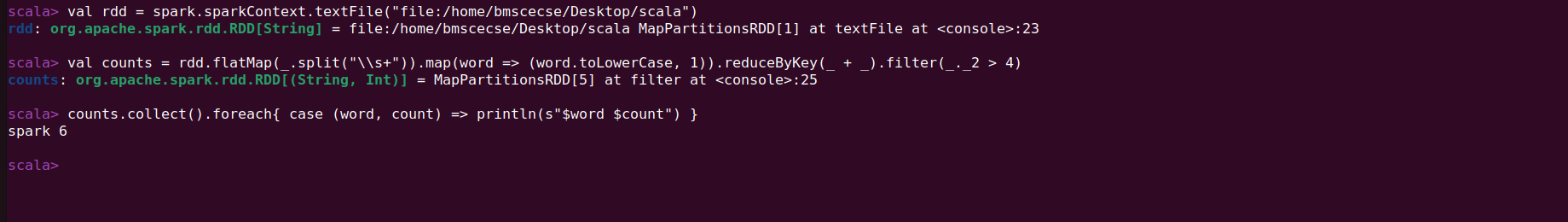




**LAB 9**

1. **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, COMMAND WITH OUTPUT



# Install NLTK and download required data (run once)

!pip install nltk

import nltk

nltk.download('punkt')

nltk.download('stopwords')

nltk.download('wordnet')

from pyspark.sql import SparkSession

from pyspark.sql.functions import col, lower, regexp\_replace, split, explode, udf

from pyspark.sql.types import ArrayType, StringType

from pyspark.ml.feature import StopWordsRemover

from nltk.stem import WordNetLemmatizer

# Initialize SparkSession

spark = SparkSession.builder.appName("TextProcessing").getOrCreate()

# Define your input lines

lines = [

    "Hello, I hate you.",

    "I hate that I love you.",

    "Don't want to, but I can't put",

    "nobody else above you."

]

# Create DataFrame from lines

df = spark.createDataFrame(lines, "string").toDF("value")

# Step 1: Lowercase and remove punctuation

df\_clean = df.select(regexp\_replace(lower(col("value")), "[^a-zA-Z\\s]", "").alias("cleaned"))

# Step 2: Tokenize the cleaned text

df\_tokens = df\_clean.select(split(col("cleaned"), "\\s+").alias("tokens"))

# Step 3: Remove stop words

remover = StopWordsRemover(inputCol="tokens", outputCol="filtered")

df\_filtered = remover.transform(df\_tokens)

# Step 4: Lemmatization using NLTK WordNetLemmatizer with UDF

lemmatizer = WordNetLemmatizer()

def lemmatize\_words(words):

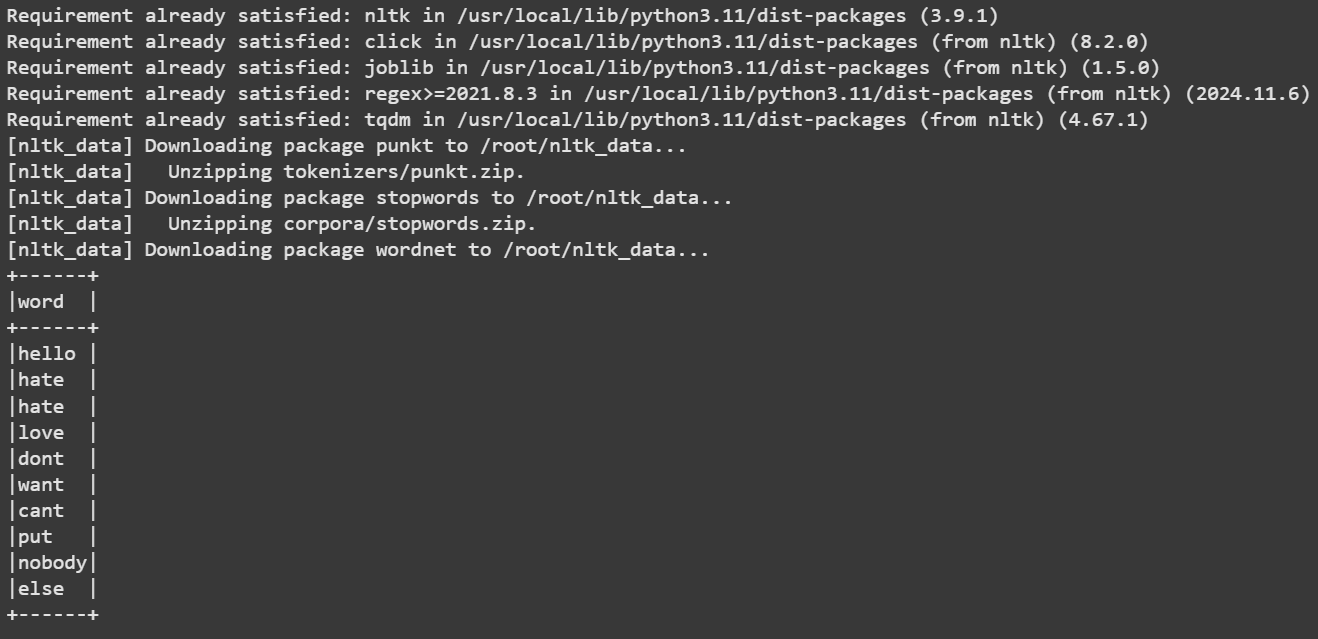
    return [lemmatizer.lemmatize(word) for word in words]

lemmatize\_udf = udf(lemmatize\_words, ArrayType(StringType()))

df\_lemmatized = df\_filtered.withColumn("lemmatized", lemmatize\_udf(col("filtered")))

# Step 5: Explode the lemmatized words and show results

df\_lemmatized.select(explode(col("lemmatized")).alias("word")).show(truncate=False)

****