**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

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



**LAB REPORT**

**on**

**BIG DATA ANALYTICS**

**(20CS6PEBDA)**

***Submitted by***

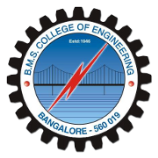
**SAURAB V MOTAGI(1BM19CS145)**

***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**” was carried out by **SAURAB V MOTAGI(1BM19CS145),** who is bona fide 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 the course **BIG DATA ANALYTICS (20CS6PEBDA)** work prescribed for the said degree.

Name of the Lab-In charge               **DR SHYAMALA**

Designation Assistant Professor

Department of CSE Department of CSE

BMSCE, Bengaluru BMSCE, Bengaluru

`

**INDEX SHEET**

|  |  |  |
| --- | --- | --- |
| **Sl.**  **No.** | **Experiment Title** | **Page No.** |
| **1** | **Cassandra Lab Program 1: - Student Database** | **3** |
| **2** | **Cassandra Lab Program 2: - Library Database** | **8** |
| **3** | **MongoDB- CRUD Demonstration** | **13** |
| **4** | **Hadoop Installation** | **28** |
| **5** | **Hadoop Commands** | **29** |
| **6** | **Hadoop Programs: Word Count** | **32** |
| **7** | **Hadoop Programs: Top N** | **36** |
| **8** | **Hadoop Programs: Average Temperature & Mean max temperature** | **40** |
| **9** | **Hadoop Programs: Join** | **49** |
| **10** | **Scala Programs: Word Count and print “Hello world”** | **62** |
| **11** | **Scala Programs: Word Count greater than 4** | **65** |

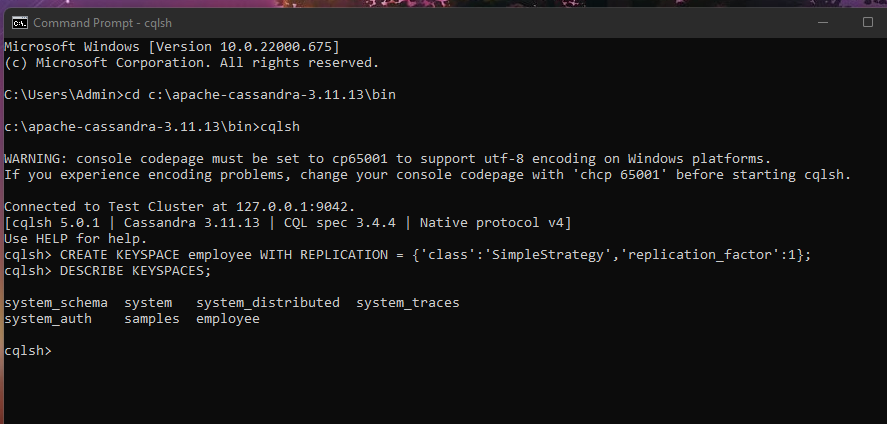
**Course Outcome**

|  |  |
| --- | --- |
| 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. **Cassandra Lab Program 1: -**

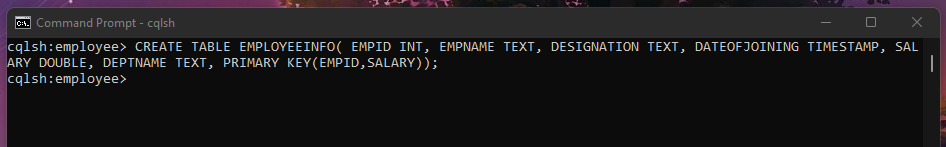
Perform the following DB operations using Cassandra.

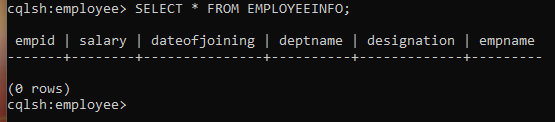
1. Create a key space by name Employee



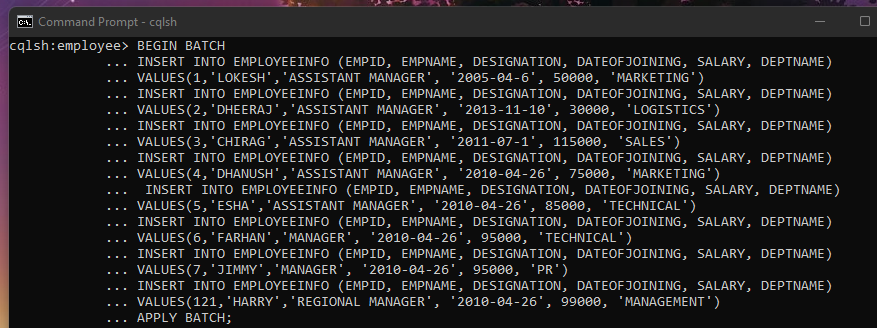
2. Create a column family by name Employee-Info with attributes Emp\_Id Primary Key, Emp\_Name,

Designation, Date\_of\_Joining, Salary, Dept\_Name



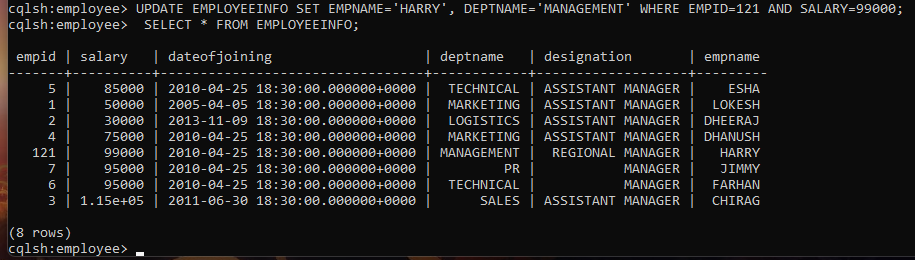


3. Insert the values into the table in batch

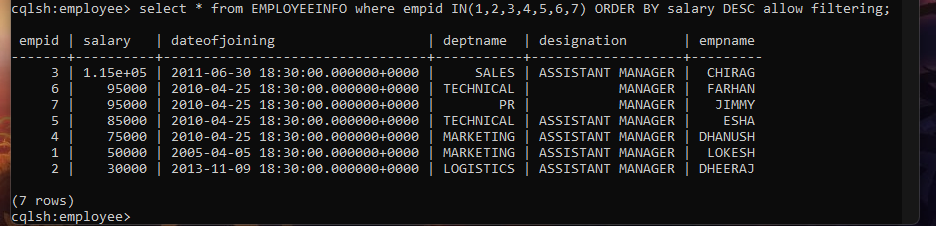




4. Update Employee name and Department of Emp-Id 121

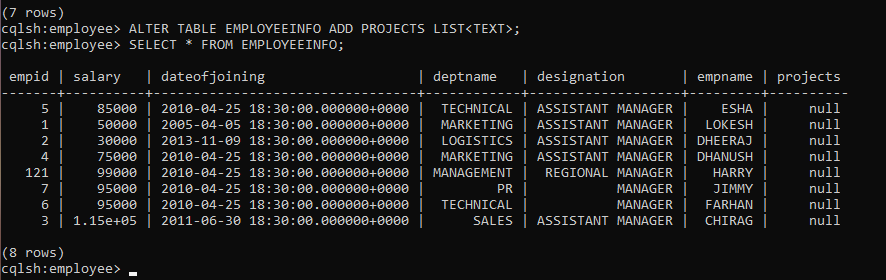


5. Sort the details of Employee records based on salary (Note:- cql>PAGING OFF)

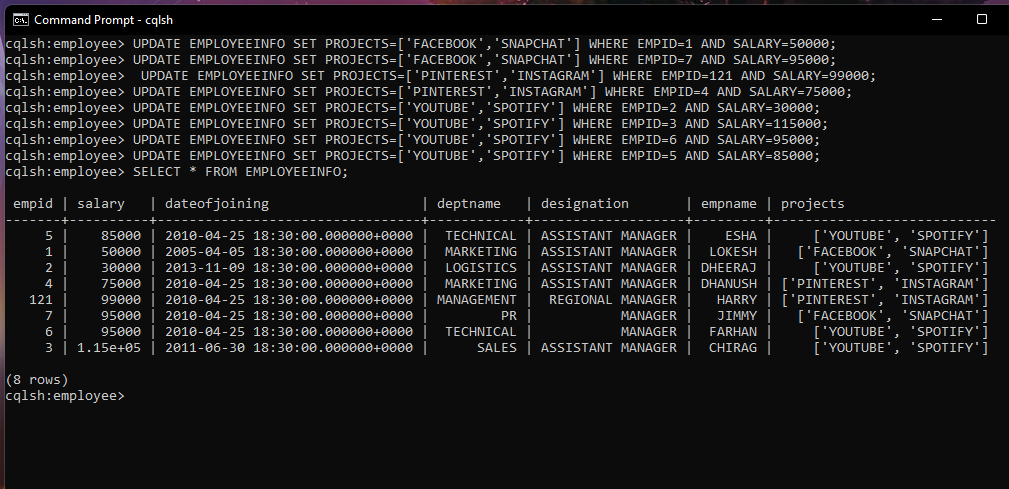


6. Alter the schema of the table Employee\_Info to add a column Projects which stores a set

of Projects done by the corresponding Employee.

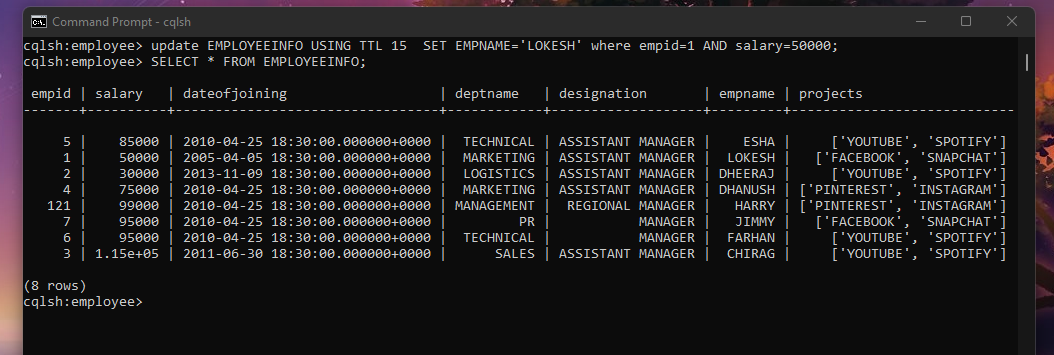


7. Update the altered table to add project names.



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

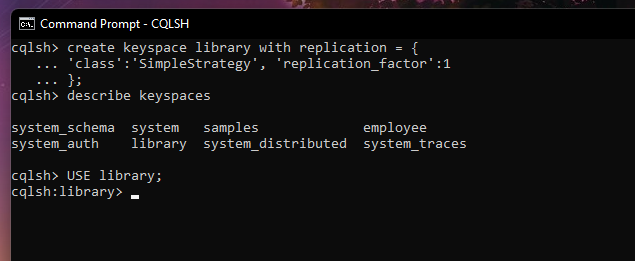
//BEFORE 15 seconds



1. **Cassandra Lab Program 2: -**

Perform the following DB operations using Cassandra.

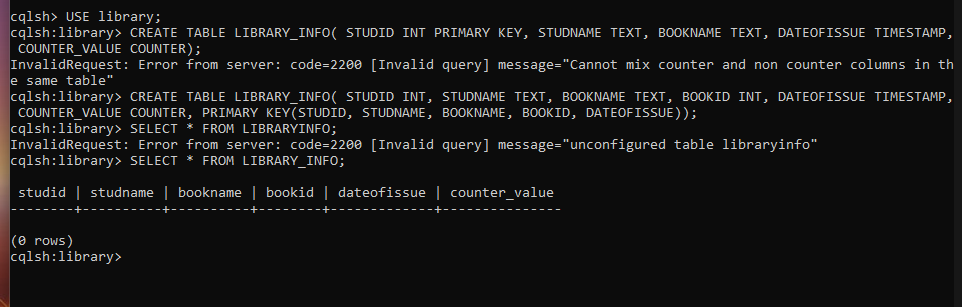
1.Create a key space by name Library



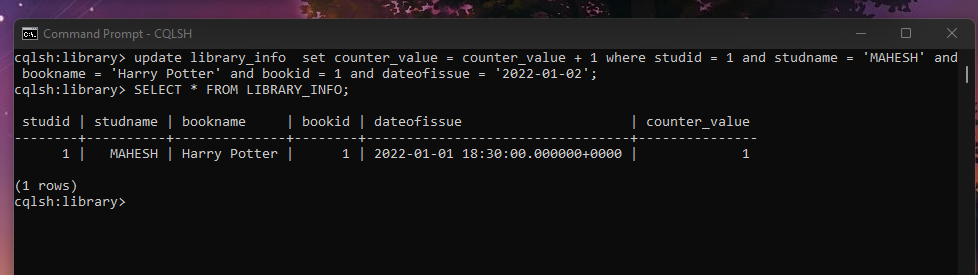
2. Create a column family by name Library-Info with attributes Stud\_Id Primary Key,

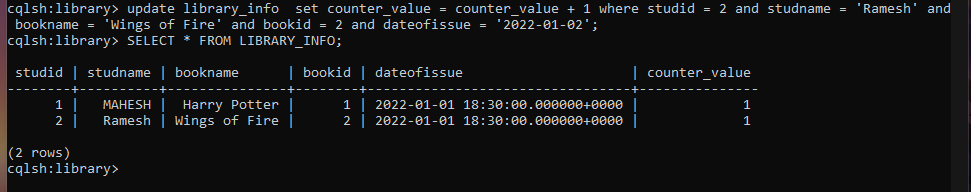
Counter\_value of type Counter,

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

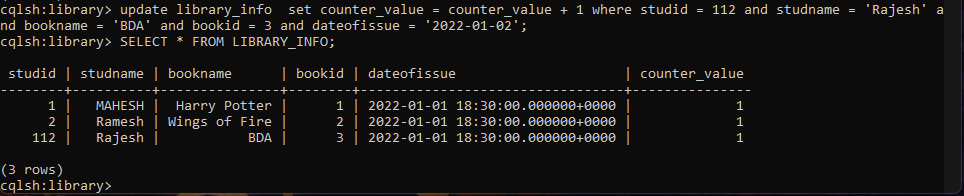


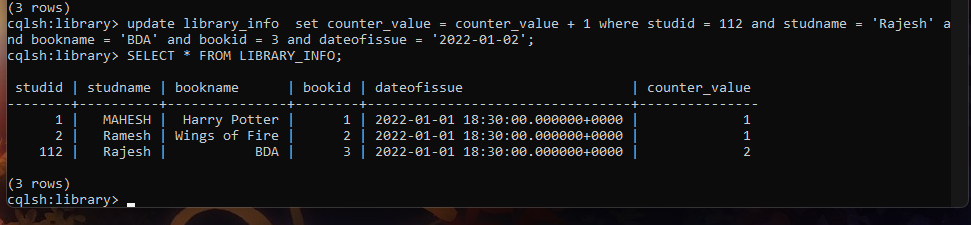
3.Insert the values into the table in batch

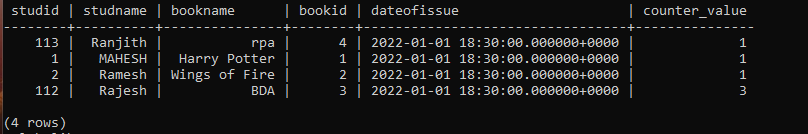




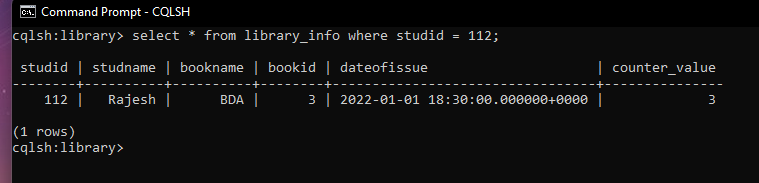
4. Display the details of the table created and increase the value of the counter



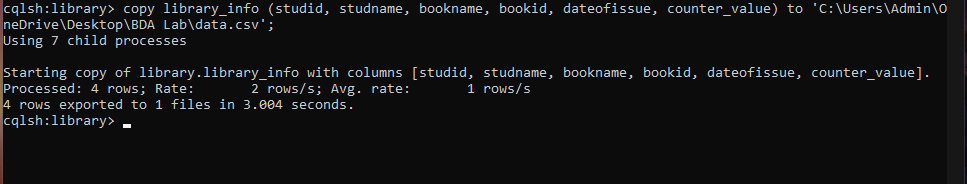


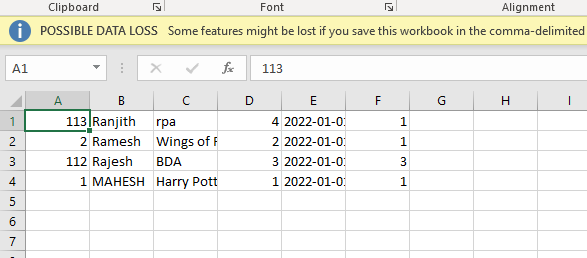


5. Write a query to show that a student with id 112 has taken a book “BDA” 3 times.

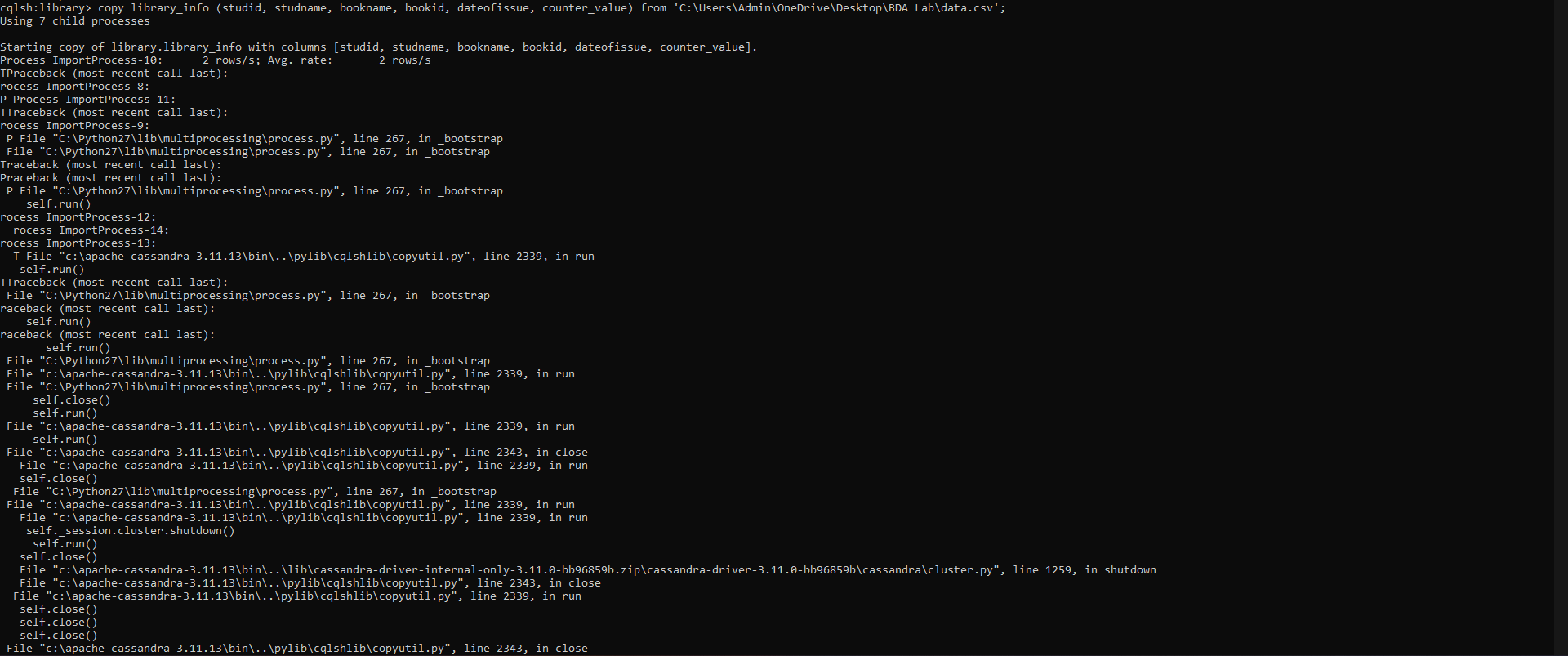


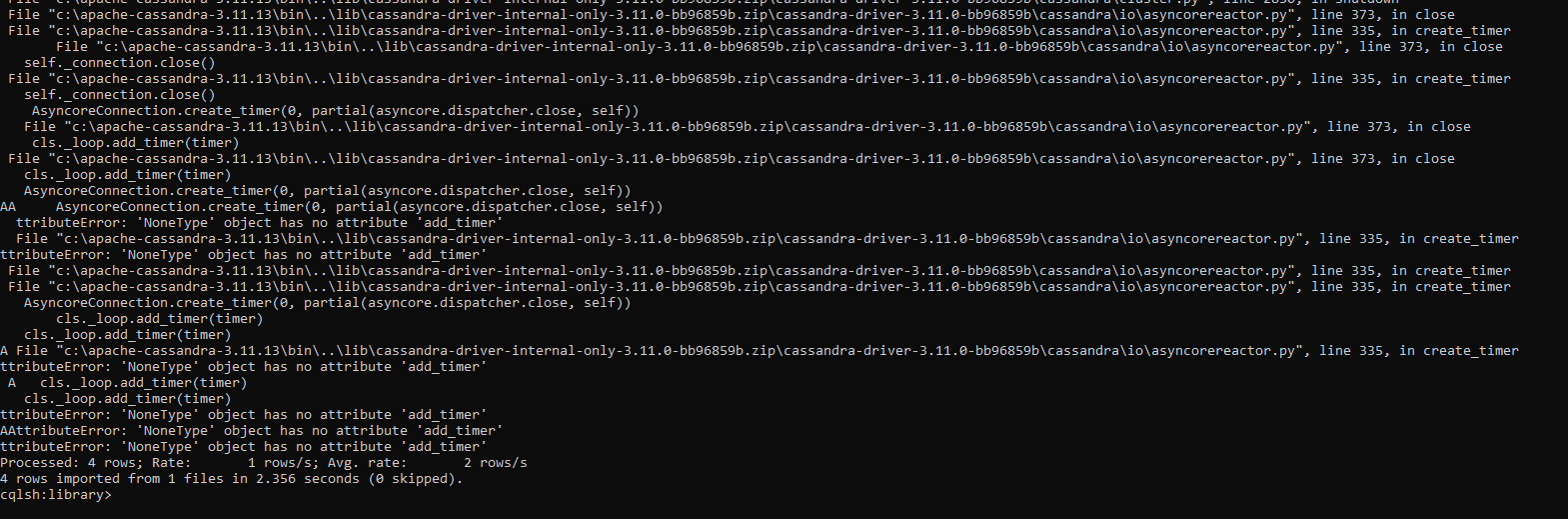
6. Export the created column to a csv file





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





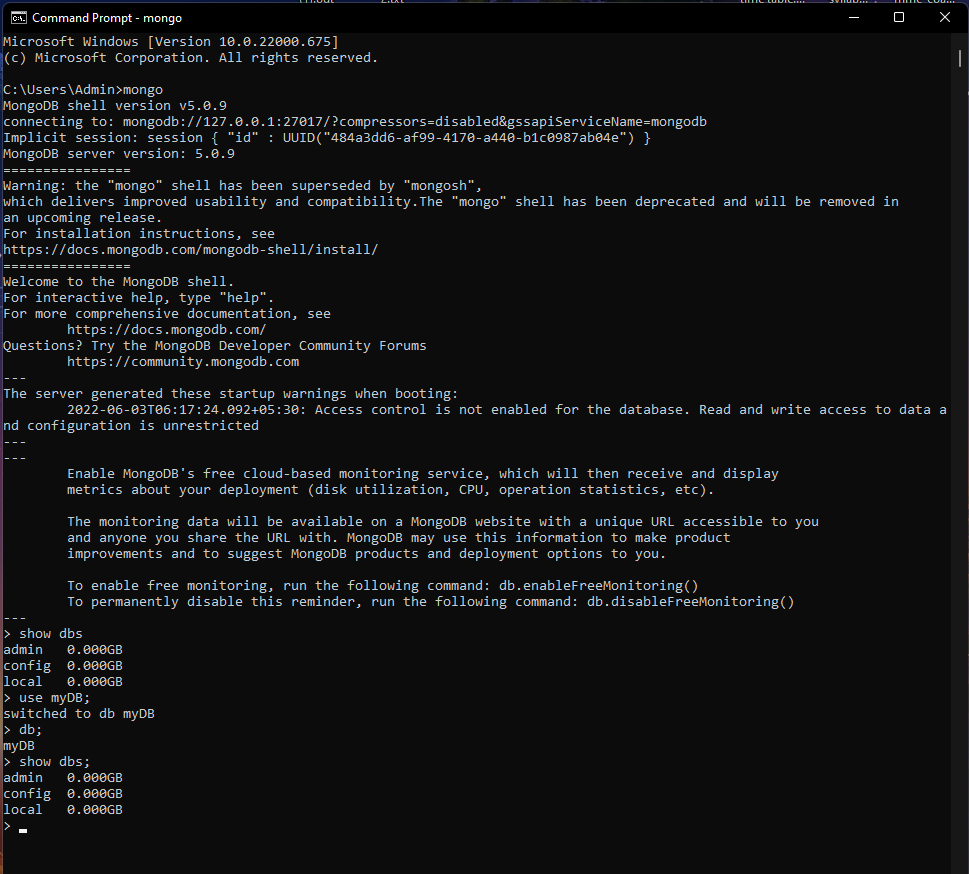
**3.a. MongoDB Lab Program 1 (CRUD Demonstration): -**

Execute the queries and upload a document with output.

I. CREATE DATABASE IN MONGODB.

use myDB;

db; (Confirm the existence of your database)

show dbs; (To list all databases) 

II.CRUD (CREATE, READ, UPDATE, DELETE) OPERATIONS

1. To create a collection by the name “Student”. Let us take a look at the collection list

prior to the creation of the new collection “Student”.

db.createCollection(“Student”); =&gt; sql equivalent CREATE TABLE STUDENT(…);

2. To drop a collection by the name “Student”.

db.Student.drop();

3. Create a collection by the name “Students” and store the following data in it.

db.Student.insert({\_id:1,StudName:&quot;MichelleJacintha&quot;,Grade:&quot;VII&quot;,Hobbies:&quot;InternetS

urfing&quot;});

4. Insert the document for “AryanDavid” in to the Students collection only if it does not

already exist in the collection. However, if it is already present in the collection, then

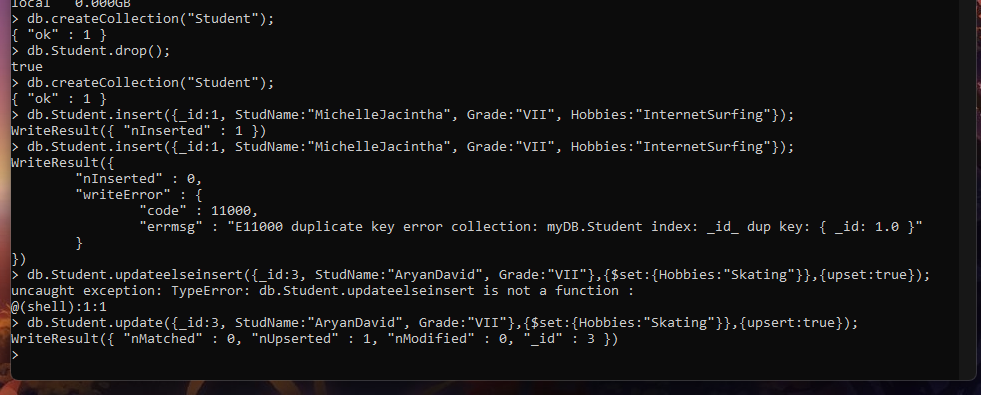
update the document with new values. (Update his Hobbies from “Skating” to “Chess”.

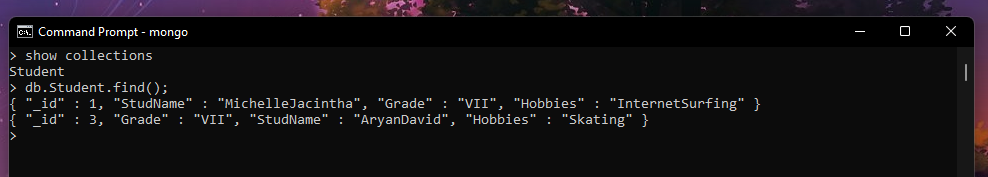
) Use “Update else insert” (if there is an existing document, it will attempt to update it,

if there is no existing document then it will insert it).

db.Student.update({\_id:3,StudName:&quot;AryanDavid&quot;,Grade:&quot;VII&quot;},{$set:{Hobbies:&quot;Skatin

g&quot;}},{upsert:true});





5. FIND METHOD

A. To search for documents from the “Students” collection based on certain search

criteria.

db.Student.find({StudName:&quot;Aryan David&quot;});

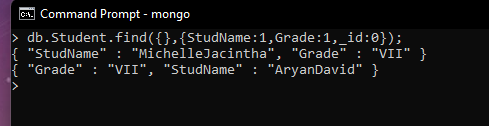
({cond..},{columns.. column:1, columnname:0} )



B. To display only the StudName and Grade from all the documents of the Students

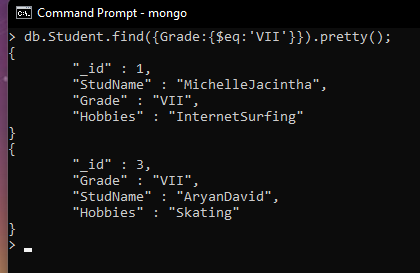
collection. The identifier\_id should be suppressed and NOT displayed.

db.Student.find({},{StudName:1,Grade:1,\_id:0});



C. To find those documents where the Grade is set to ‘VII’

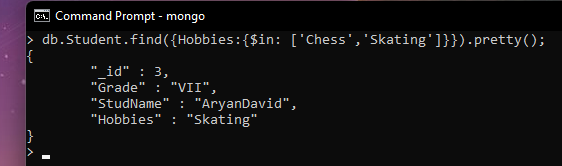
db.Student.find({Grade:{$eq:&#39;VII&#39;}}).pretty();



D. To find those documents from the Students collection where the Hobbies is set to

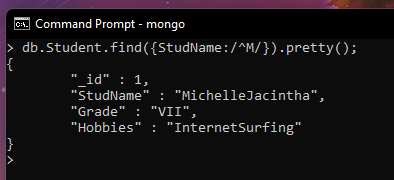
either ‘Chess’ or is set to ‘Skating’.

db.Student.find({Hobbies :{ $in: [&#39;Chess&#39;,&#39;Skating&#39;]}}).pretty ();



E. To find documents from the Students collection where the StudName begins with “M”.

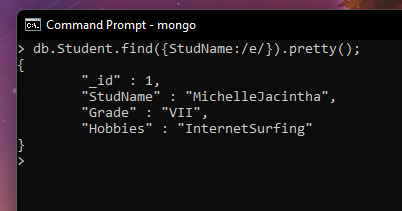
db.Student.find({StudName:/^M/}).pretty();



F. To find documents from the Students collection where the StudNamehas an “e” in any

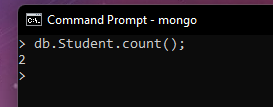
position.

db.Student.find({StudName:/e/}).pretty();



G. To find the number of documents in the Students collection.

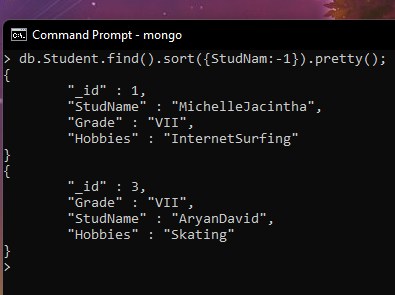
db.Student.count();



H. To sort the documents from the Students collection in the descending order of

StudName.

db.Student.find().sort({StudName:-1}).pretty();



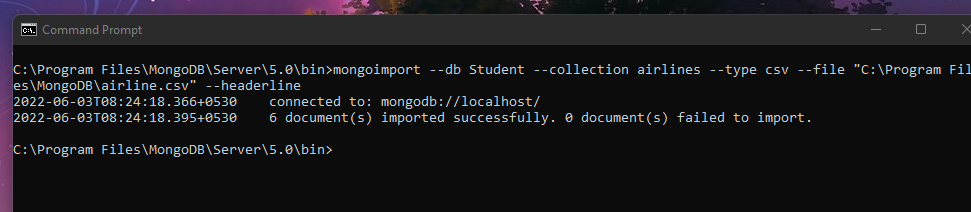
III. Import data from a CSV file

Given a CSV file “sample.txt” in the D:drive, import the file into the MongoDB

collection, “SampleJSON”. The collection is in the database “test”.

mongoimport --db Student --collection airlines --type csv –headerline --file

/home/hduser/Desktop/airline.csv



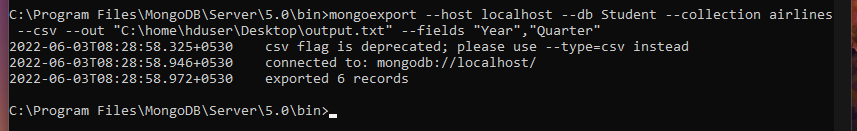
IV. Export data to a CSV file

This command used at the command prompt exports MongoDB JSON documents from

“Customers” collection in the “test” database into a CSV file “Output.txt” in the D:drive.

mongoexport --host localhost --db Student --collection airlines --csv --out

/home/hduser/Desktop/output.txt –fields “Year”,”Quarter”

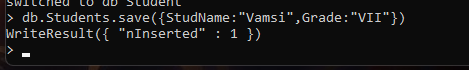


V. Save Method :

Save() method will insert a new document, if the document with the \_id does not

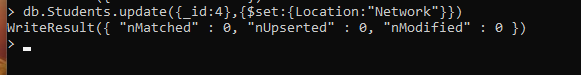
exist. If it exists it will replace the exisiting document.

db.Students.save({StudName:”Vamsi”, Grade:”VI”})



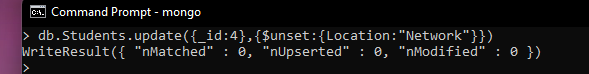
VI. Add a new field to existing Document:

db.Students.update({\_id:4},{$set:{Location:”Network”}})



VII. Remove the field in an existing Document

db.Students.update({\_id:4},{$unset:{Location:”Network”}})



VIII. Finding Document based on search criteria suppressing few fields

db.Student.find({\_id:1},{StudName:1,Grade:1,\_id:0});

To find those documents where the Grade is not set to ‘VII’

db.Student.find({Grade:{$ne:&#39;VII&#39;}}).pretty();

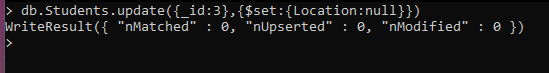
To find documents from the Students collection where the StudName ends with s.

db.Student.find({StudName:/s$/}).pretty();





IX. to set a particular field value to NULL



X Count the number of documents in Student Collections



XI. Count the number of documents in Student Collections with grade :VII

db.Students.count({Grade:”VII”})

retrieve first 3 documents

db.Students.find({Grade:”VII”}).limit(3).pretty();

Sort the document in Ascending order

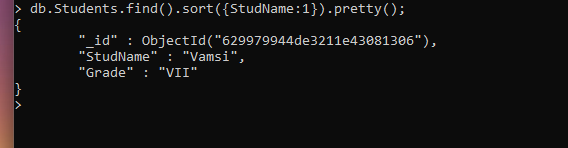
db.Students.find().sort({StudName:1}).pretty();

Note:

for desending order : db.Students.find().sort({StudName:-1}).pretty();

to Skip the 1 st two documents from the Students Collections

db.Students.find().skip(2).pretty()

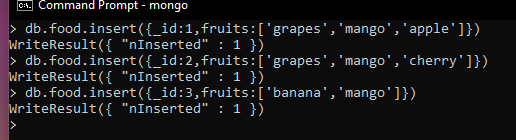


XII. Create a collection by name “food” and add to each document add a “fruits” array

db.food.insert( { \_id:1, fruits:[&#39;grapes&#39;,&#39;mango&#39;,&#39;apple&#39;] } )

db.food.insert( { \_id:2, fruits:[&#39;grapes&#39;,&#39;mango&#39;,&#39;cherry&#39;] } )

db.food.insert( { \_id:3, fruits:[&#39;banana&#39;,&#39;mango&#39;] } )



To find those documents from the “food” collection which has the “fruits array”

constitute of “grapes”, “mango” and “apple”.

db.food.find ( {fruits: [&#39;grapes&#39;,&#39;mango&#39;,&#39;apple&#39;] } ). pretty().



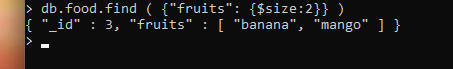
To find in “fruits” array having “mango” in the first index position.

db.food.find ( {&#39;fruits.1&#39;:&#39;grapes&#39;} )



To find those documents from the “food” collection where the size of the array is two.

db.food.find ( {“fruits”: {$size:2}} )



To find the document with a particular id and display the first two elements from the

array “fruits”

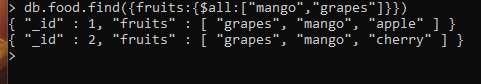
db.food.find({\_id:1},{“fruits”:{$slice:2}})



To find all the documets from the food collection which have elements mango and

grapes in the array “fruits”

db.food.find({fruits:{$all:[“mango”,”grapes”]}})



update on Array:

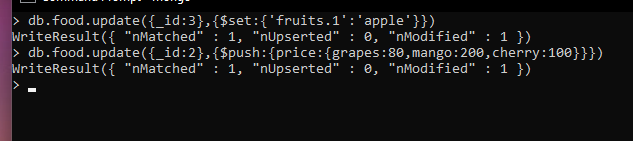
using particular id replace the element present in the 1 st index position of the fruits

array with apple

db.food.update({\_id:3},{$set:{&#39;fruits.1&#39;:&#39;apple&#39;}})

insert new key value pairs in the fruits array

db.food.update({\_id:2},{$push:{price:{grapes:80,mango:200,cherry:100}}})



Note: perform query operations using - pop, addToSet, pullAll and pull

XII. Aggregate Function :

Create a collection Customers with fields custID, AcctBal, AcctType.

Now group on “custID” and compute the sum of “AccBal”.

db.Customers.aggregate ( {$group : { \_id : “$custID”,TotAccBal : {$sum:”$AccBal”} } } );

match on AcctType:”S” then group on “CustID” and compute the sum of “AccBal”.

db.Customers.aggregate ( {$match:{AcctType:”S”}},{$group : { \_id : “$custID”,TotAccBal :

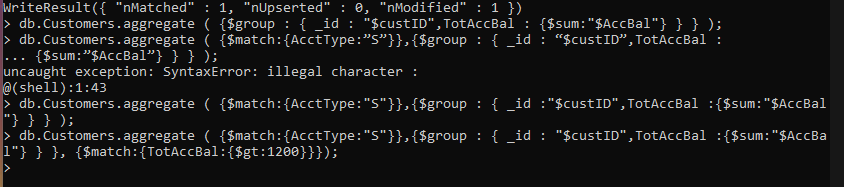
{$sum:”$AccBal”} } } );

match on AcctType:”S” then group on “CustID” and compute the sum of “AccBal” and

total balance greater than 1200.

db.Customers.aggregate ( {$match:{AcctType:”S”}},{$group : { \_id : “$custID”,TotAccBal :

{$sum:”$AccBal”} } }, {$match:{TotAccBal:{$gt:1200}}});



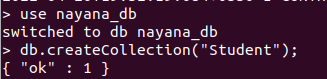
**3.b.MongoDB Lab Program 2 (CRUD Demonstration): -**

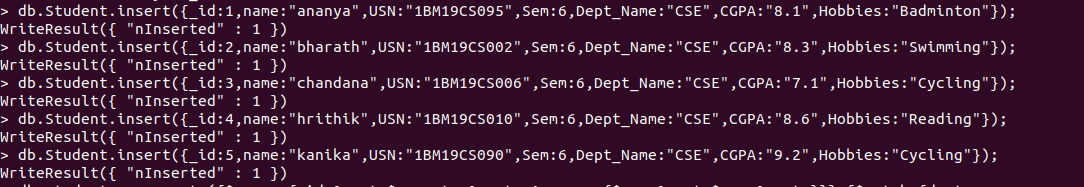
1) Using MongoDB

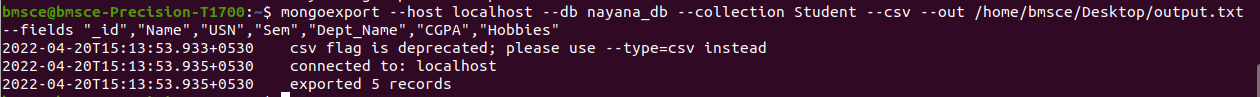
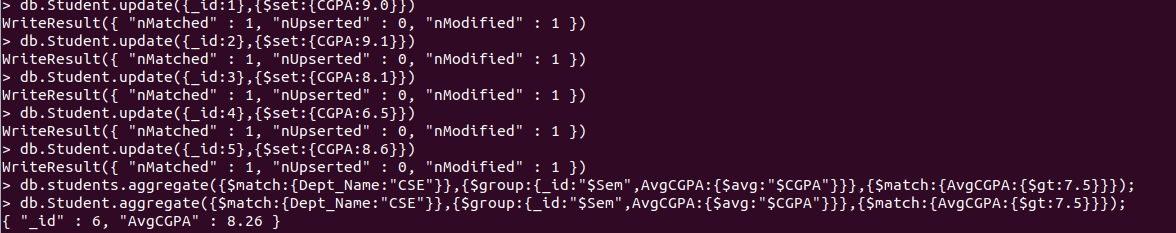
1. Create a database for Students and Create a Student Collection (\_id,Name, USN, Semester, Dept\_Name, CGPA, Hobbies(Set)).
2. Insert required documents to the collection.
3. First Filter on “Dept\_Name:CSE” and then group it on “Semester” and

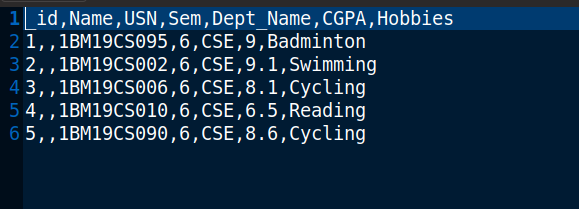
compute the Average CPGA for that semester and flter those documents where the “Avg\_CPGA” is greater than 7.5.

1. Command used to export MongoDB JSON documents from “Student” Collection into the “Students” database into a CSV fle “Output.txt”.



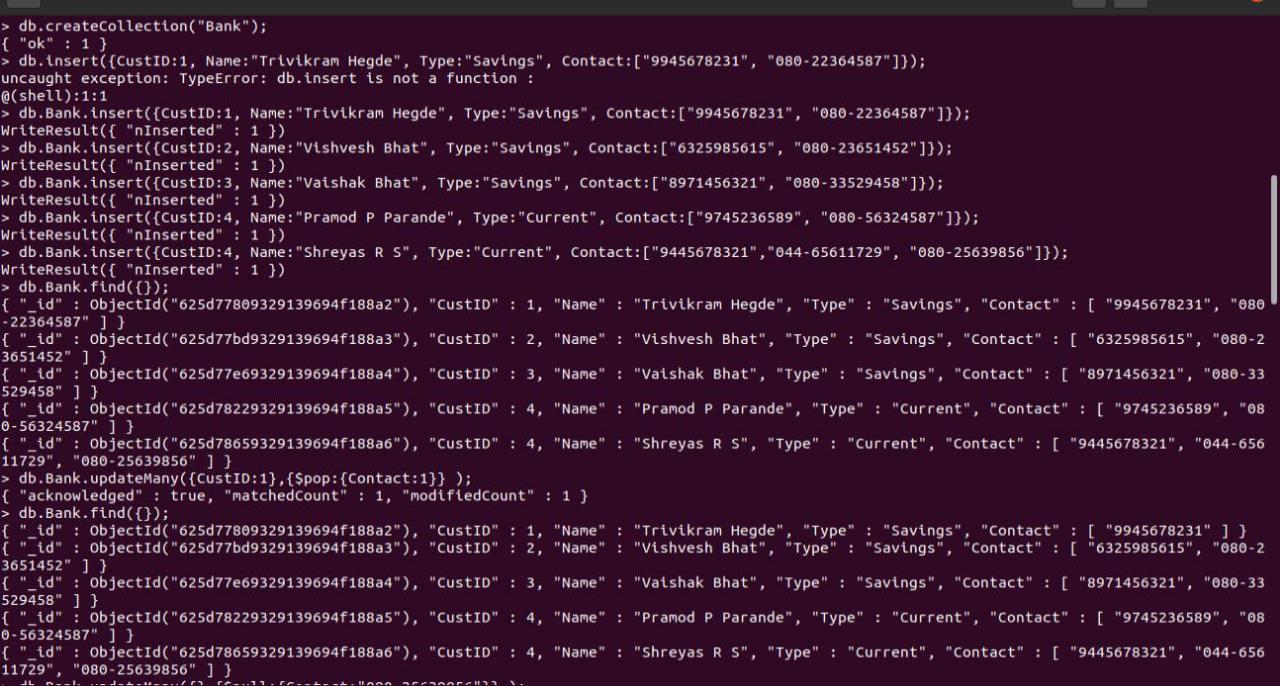


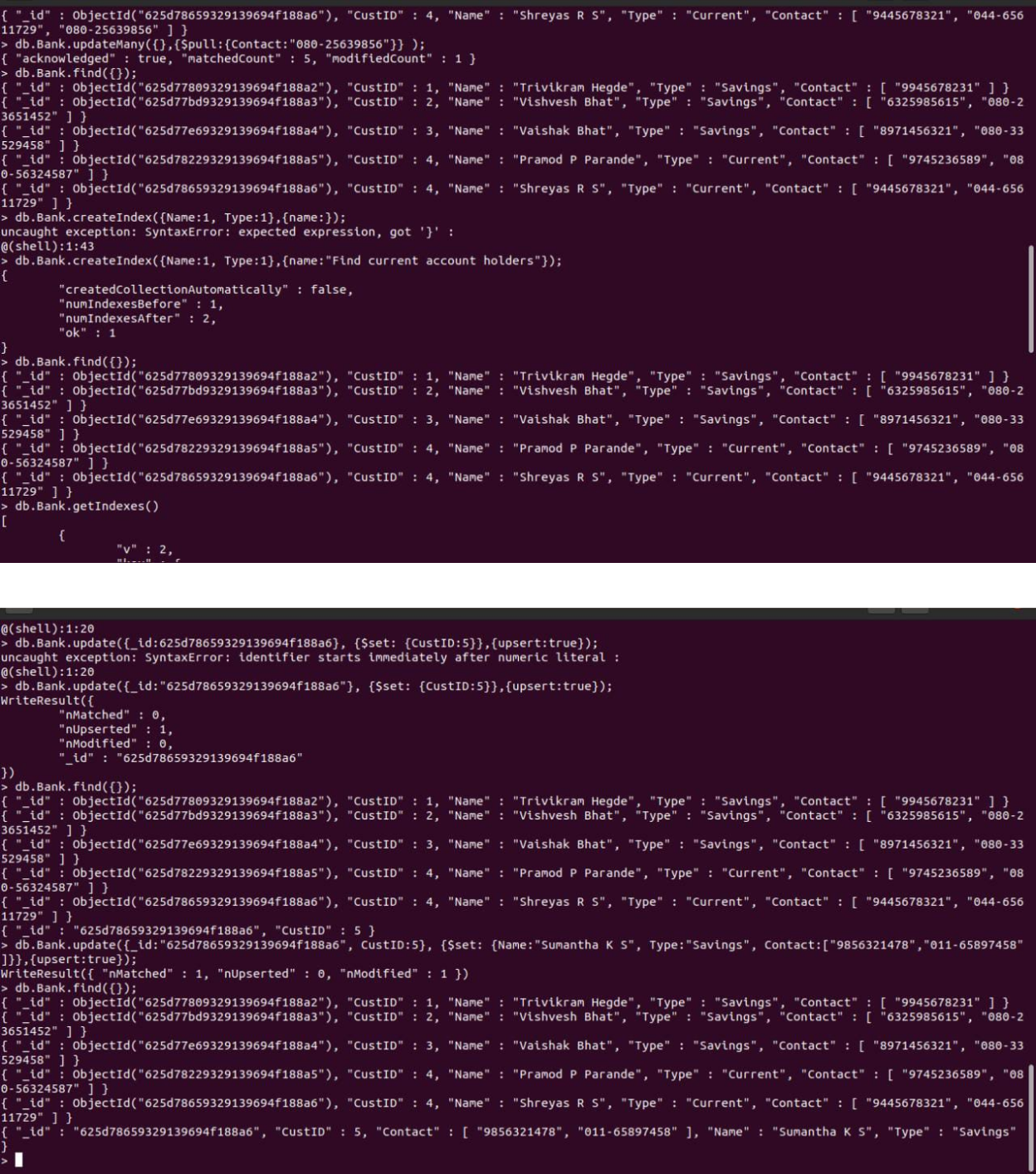




2)Create a mongodb collection Bank. Demonstrate the following by choosing felds of your choice.

1. Insert three documents
2. Use Arrays(Use Pull and Pop operation)
3. Use Index
4. Use Cursors
5. Updation





1) Using MongoDB,

1. Create a database for Faculty and Create a Faculty Collection(Faculty\_id, Name, Designation ,Department, Age, Salary, Specialization(Set)).
2. Insert required documents to the collection.
3. First Filter on “Dept\_Name:MECH” and then group it on “Designation” and

compute the Average Salary for that Designation and flter those documents where the “Avg\_Sal” is greater than 650000. iv) Demonstrate usage of import and export commands

Write MongoDB queries for the following:

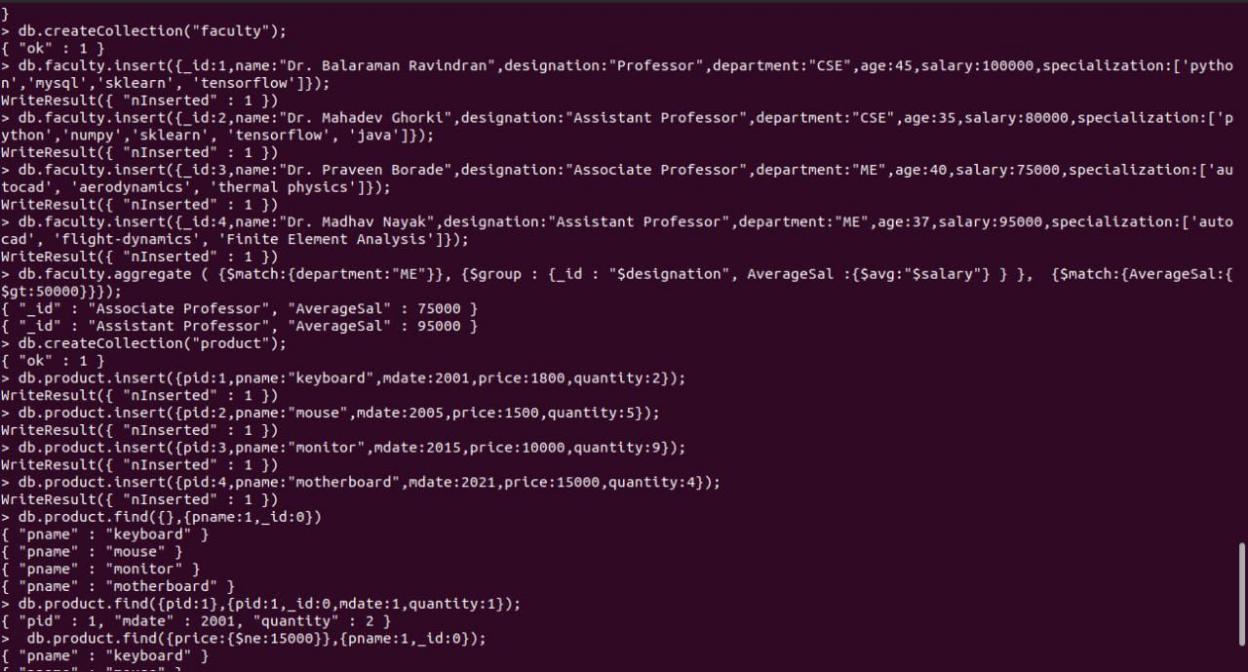
1)To display only the product name from all the documents of the product collection.

2)To display only the Product ID, ExpiryDate as well as the quantity from the document of the product collection where the \_id column is 1.

3)To fnd those documents where the price is not set to 15000.

4)To fnd those documents from the Product collection where the quantity is set to 9 and the product name is set to ‘monitor’.

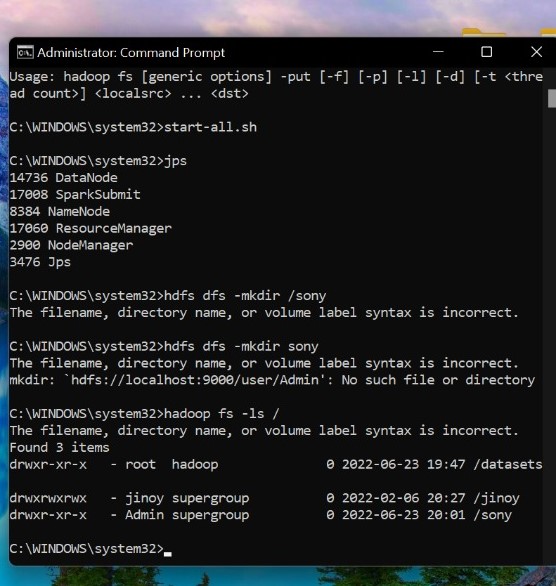
5)To fnd documents from the Product collection where the Product name ends in ‘d’.



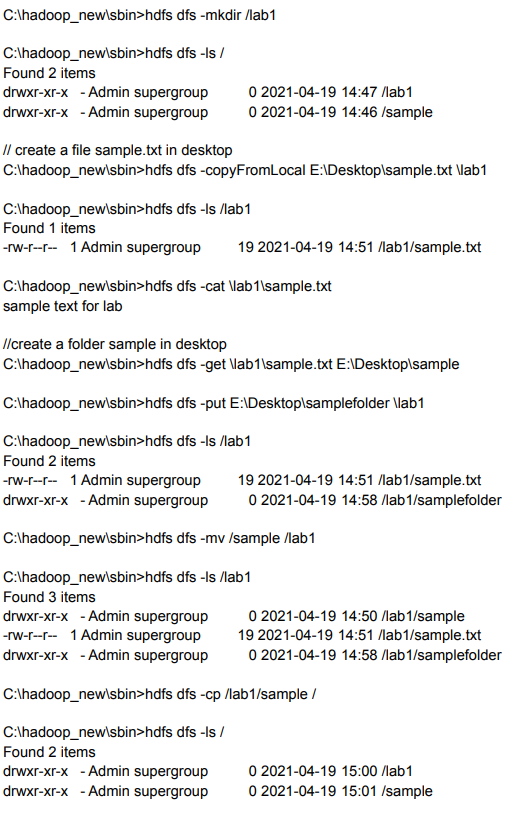
|  |  |
| --- | --- |
| 3)Create a mongodb collection Hospital. Demonstrate the following by choosing felds of | |
| choice. | |
| 1. | Insert three documents |
| 2. | Use Arrays(Use Pull and Pop operation) |
| 3. | Use Index |
| 4. | Use Cursors |
| 5. | Updation |

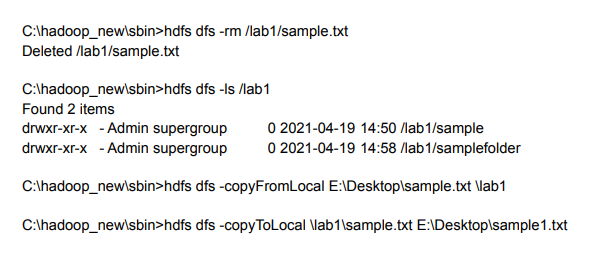


**4. Hadoop Installation(screenshot)**



1. **Hadoop Commands**





1. **Hadoop Programs: Word Count**

Create Three Java Classes into the project. Name them WCDriver(having the main function), WCMapper, WCReducer.

// Importing libraries

import java.io.IOException;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapred.MapReduceBase;

import org.apache.hadoop.mapred.Mapper;

import org.apache.hadoop.mapred.OutputCollector;

import org.apache.hadoop.mapred.Reporter;

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

// Map function

public void map(LongWritable key, Text value, OutputCollector<Text,

IntWritable> output, Reporter rep) throws IOException

{

String line = value.toString();

// Splitting the line on spaces

for (String word : line.split(" "))

{

if (word.length() > 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<Text,IntWritable, Text, IntWritable> {

// Reduce function

public void reduce(Text key, Iterator<IntWritable> value,

OutputCollector<Text, IntWritable> 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 < 2)

{

System.out.println("Please give valid inputs");

return -1;

}

JobConf conf = new JobConf(WCDriver.class);

FileInputFormat.setInputPaths(conf, new Path(args[0]));

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

conf.setMapperClass(WCMapper.class);

conf.setReducerClass(WCReducer.class);

conf.setMapOutputKeyClass(Text.class);

conf.setMapOutputValueClass(IntWritable.class);

conf.setOutputKeyClass(Text.class);

conf.setOutputValueClass(IntWritable.class);

JobClient.runJob(conf);

return 0;

}

// Main Method

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

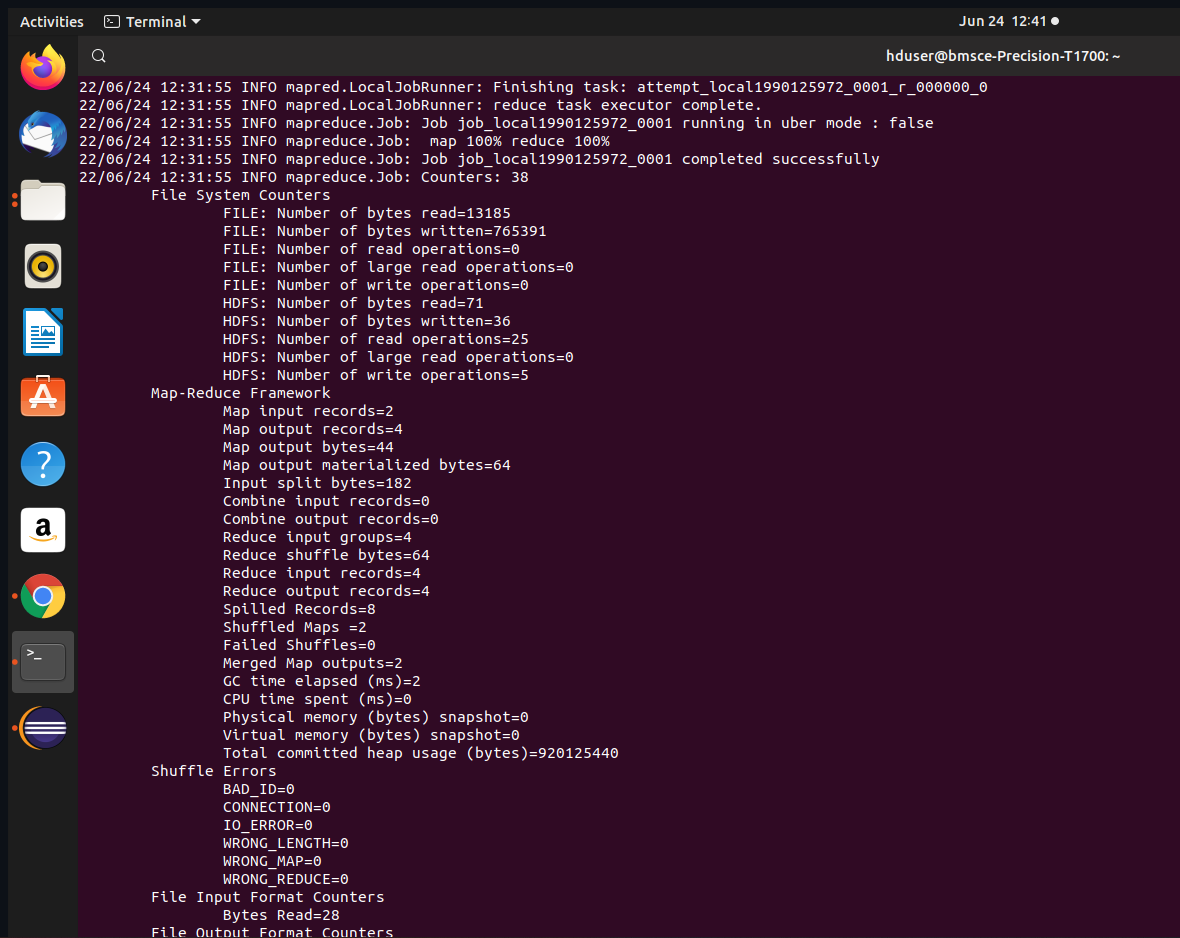
{

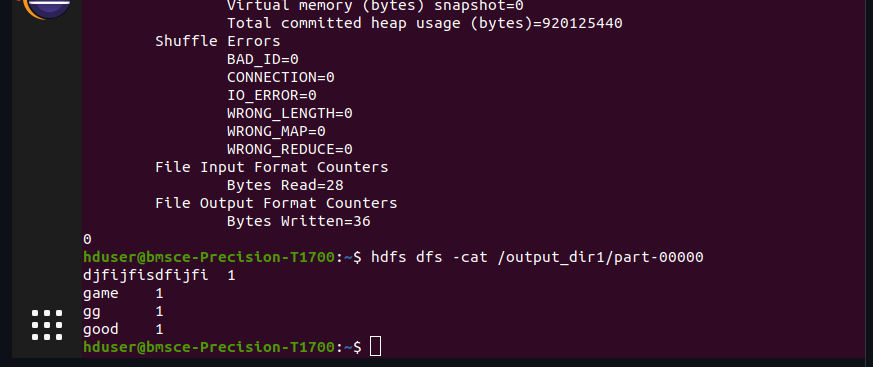
int exitCode = ToolRunner.run(new WCDriver(), args);

System.out.println(exitCode);

}

}





1. **Hadoop Programs : Top N**

Driver-TopN.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);  
 }   
 }  
 }  
}

TopNCombiner.class

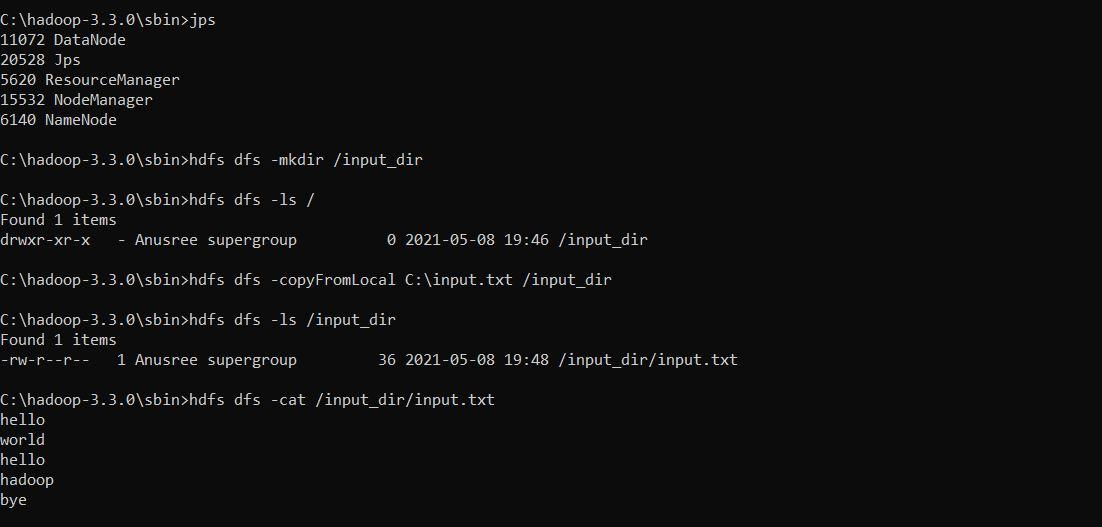
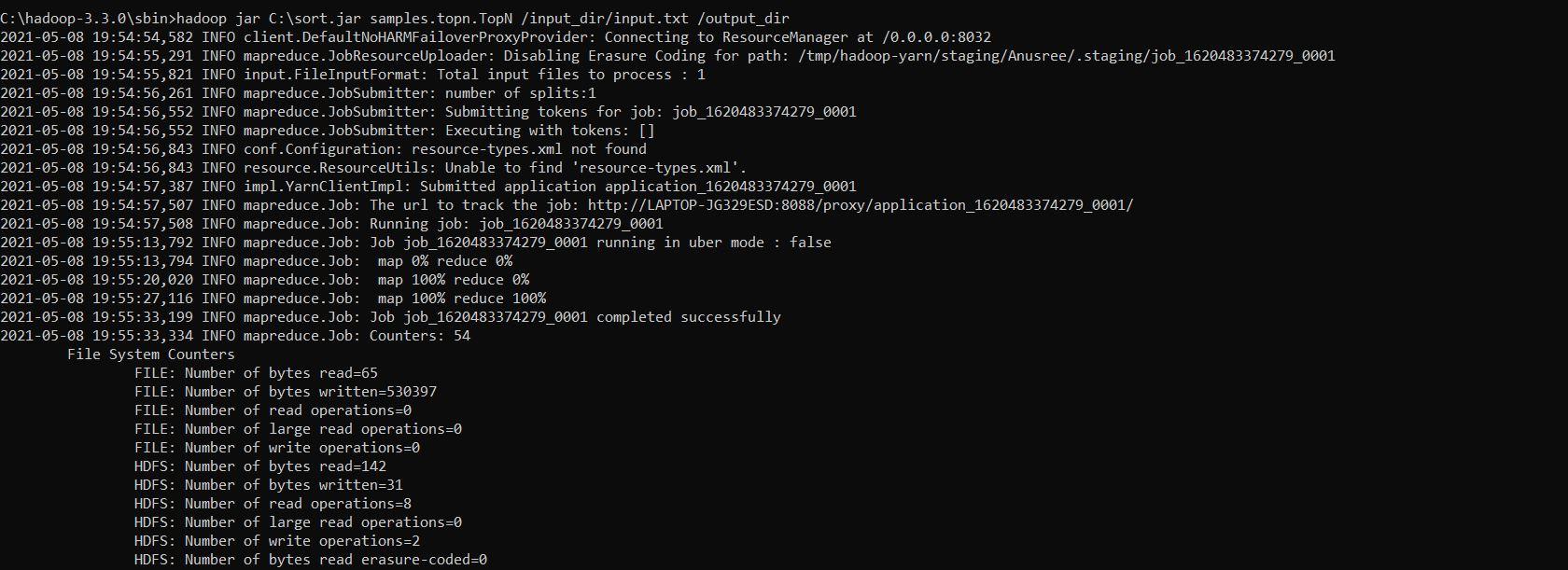
**package** samples.topn;  
  
**import** java.io.IOException;  
**import** org.apache.hadoop.io.IntWritable;  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Reducer;  
  
**public** **class** TopNCombiner **extends** Reducer<Text, IntWritable, Text, IntWritable> {  
 **public** **void** reduce(Text key, Iterable<IntWritable> values, Reducer<Text, IntWritable, Text, IntWritable>.Context context) **throws** IOException, InterruptedException {  
 **int** sum = 0;  
 **for** (IntWritable val : values)  
 sum += val.get();   
 context.write(key, **new** IntWritable(sum));  
 }  
}

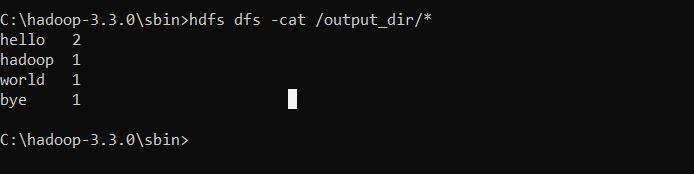
TopNMapper.class

**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);  
 }   
 }  
}

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<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));  
 }   
 }  
}



1. **Hadoop Programs : Avg Temperature and**
2. Avg temp:

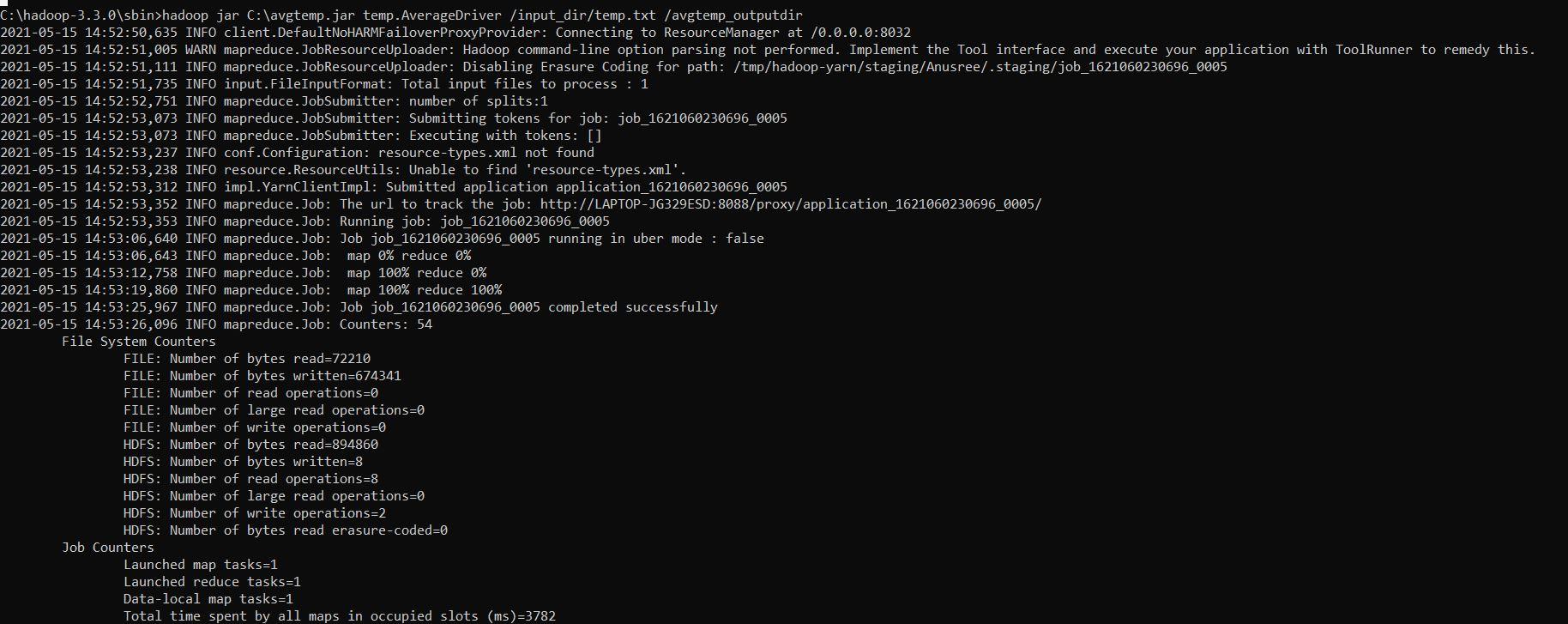
AverageDriver

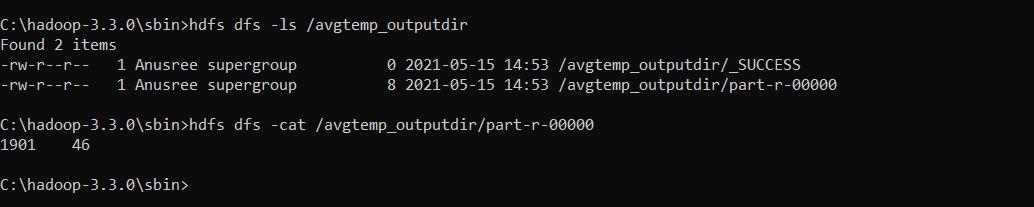
**package** temp;  
  
**import** org.apache.hadoop.fs.Path;  
**import** org.apache.hadoop.io.IntWritable;  
**import** org.apache.hadoop.io.Text;  
**import** org.apache.hadoop.mapreduce.Job;  
**import** org.apache.hadoop.mapreduce.lib.input.FileInputFormat;  
**import** org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;  
  
**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<LongWritable, Text, Text, IntWritable> {  
 **public** **static** **final** **int** MISSING = 9999;  
   
 **public** **void** map(LongWritable key, Text value, Mapper<LongWritable, Text, Text, IntWritable>.Context context) **throws** IOException, InterruptedException {  
 **int** temperature;  
 String line = value.toString();  
 String year = line.substring(15, 19);  
 **if** (line.charAt(87) == '+') {  
 temperature = Integer.parseInt(line.substring(88, 92));  
 } **else** {  
 temperature = Integer.parseInt(line.substring(87, 92));  
 }   
 String quality = line.substring(92, 93);  
 **if** (temperature != 9999 && quality.matches("[01459]"))  
 context.write(**new** Text(year), **new** IntWritable(temperature));   
 }  
}

AverageReducer

**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));  
 }  
}  




1. Mean max temp:

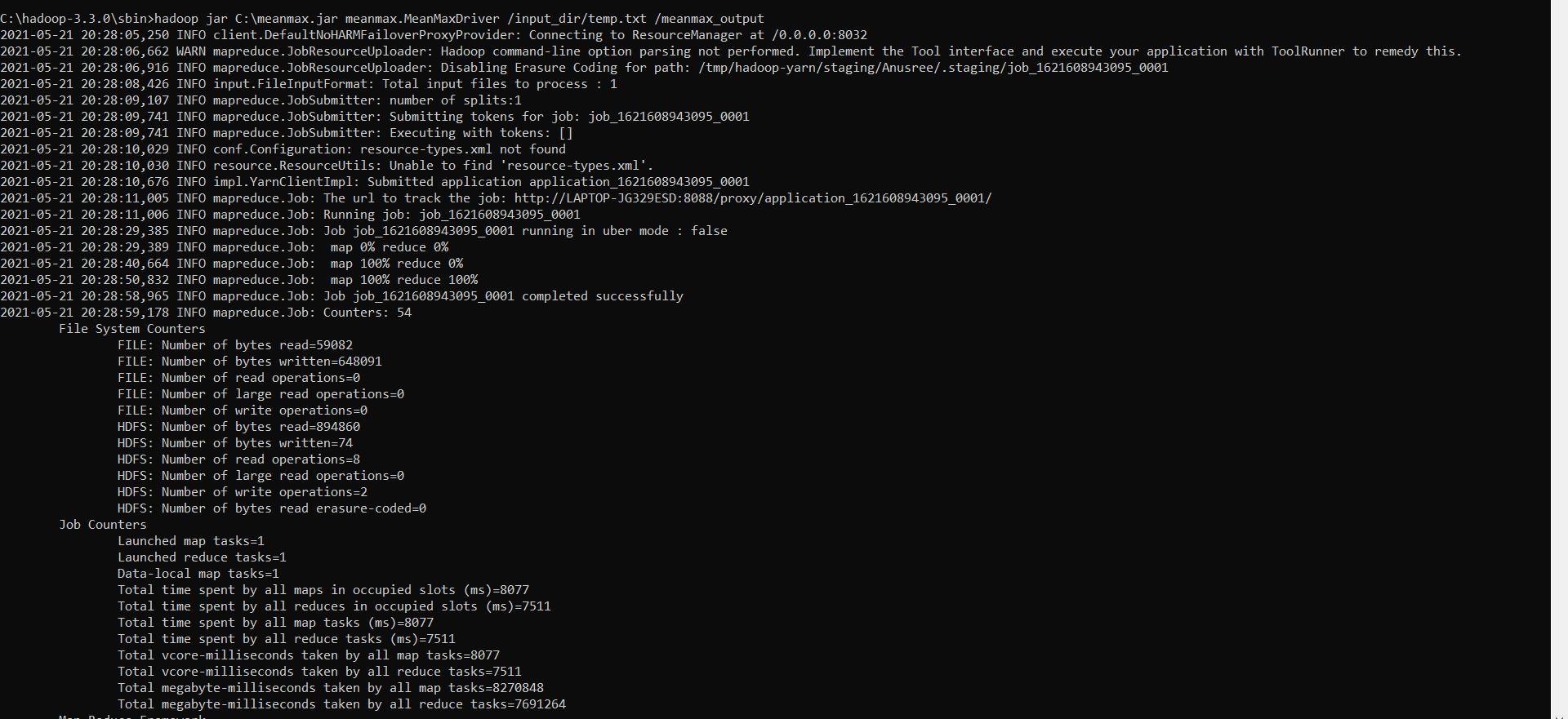
MeanMax

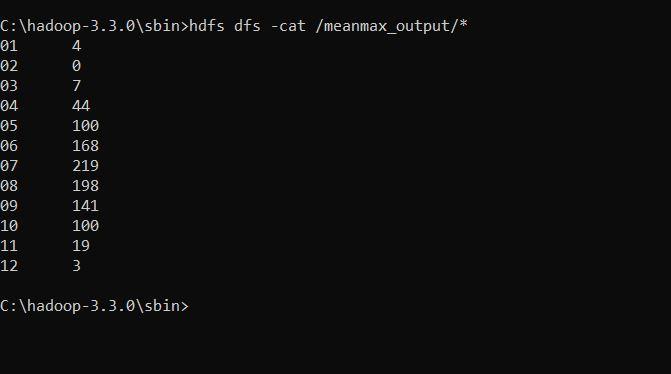
MeanMaxDriver.class

**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<LongWritable, Text, Text, IntWritable> {  
 **public** **static** **final** **int** MISSING = 9999;  
   
 **public** **void** map(LongWritable key, Text value, Mapper<LongWritable, Text, Text, IntWritable>.Context context) **throws** IOException, InterruptedException {  
 **int** temperature;  
 String line = value.toString();  
 String month = line.substring(19, 21);  
 **if** (line.charAt(87) == '+') {  
 temperature = Integer.parseInt(line.substring(88, 92));  
 } **else** {  
 temperature = Integer.parseInt(line.substring(87, 92));  
 }   
 String quality = line.substring(92, 93);  
 **if** (temperature != 9999 && quality.matches("[01459]"))  
 context.write(**new** Text(month), **new** IntWritable(temperature));   
 }  
}  
  
MeanMaxReducer.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));  
 }  
}





1. **Hadoop Programs : Join**

// 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) {

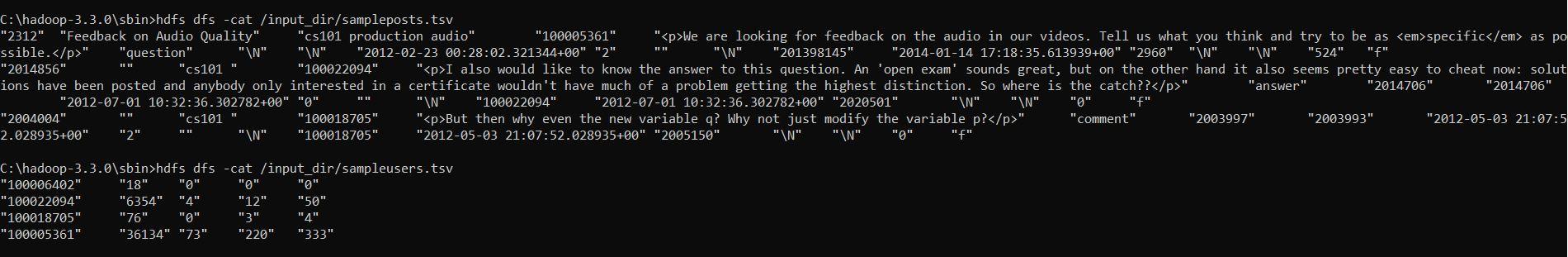
return ((TextPair) a).first.compareTo(((TextPair) b).first);

}

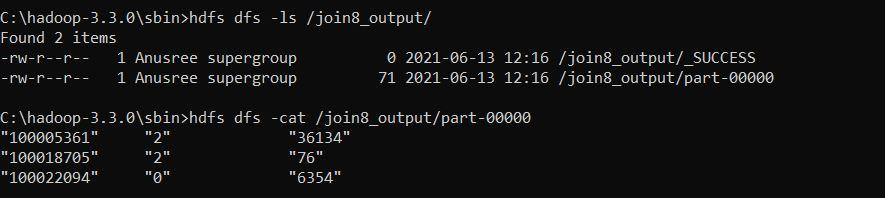
return super.compare(a, b);

}

} }



LAB8/Department\_Employee\_join\_example/DeptName.txt



1. **Scala Programs: Word Count and hello world**

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

data.collect;

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

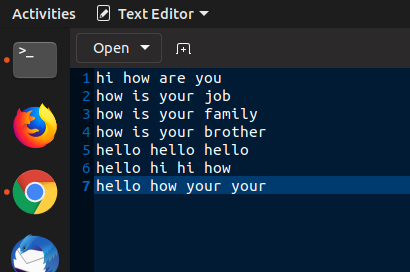
splitdata.collect;

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

mapdata.collect;

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

reducedata.collect;





$scala

scala> println("Hello World!");

Hello World!

1. **Scala Programs: Word Count Greater Than 4**

val textFile = sc.textFile("/home/admin/Desktop/wc.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):\_\*)// sort in descending order based on values

println(sorted)

for((k,v)<-sorted)

{

  if(v>4)

    {

     print(k+",")

       print(v)

       println()

     }

}

