**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

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

****

**LAB REPORT**

**on**

**Big Data and Analytics**

(22CS6PEBDA)

***Submitted by***

**Aditi Raghunandan (1BM21CS005)**

***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-2024 to July-2024**

**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 and Analytics (22CS6PEBDA)**” carried out by **Aditi Raghunandan (1BM21CS005),** 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 2024. The Lab report has been approved as it satisfies the academic requirements in respect of a **Course Title - (Course code)** work prescribed for the said degree.

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

Assistant Professor Professor and Head

Department of CSE Department of CSE

BMSCE, Bengaluru BMSCE, Bengaluru

`

**Index Sheet**

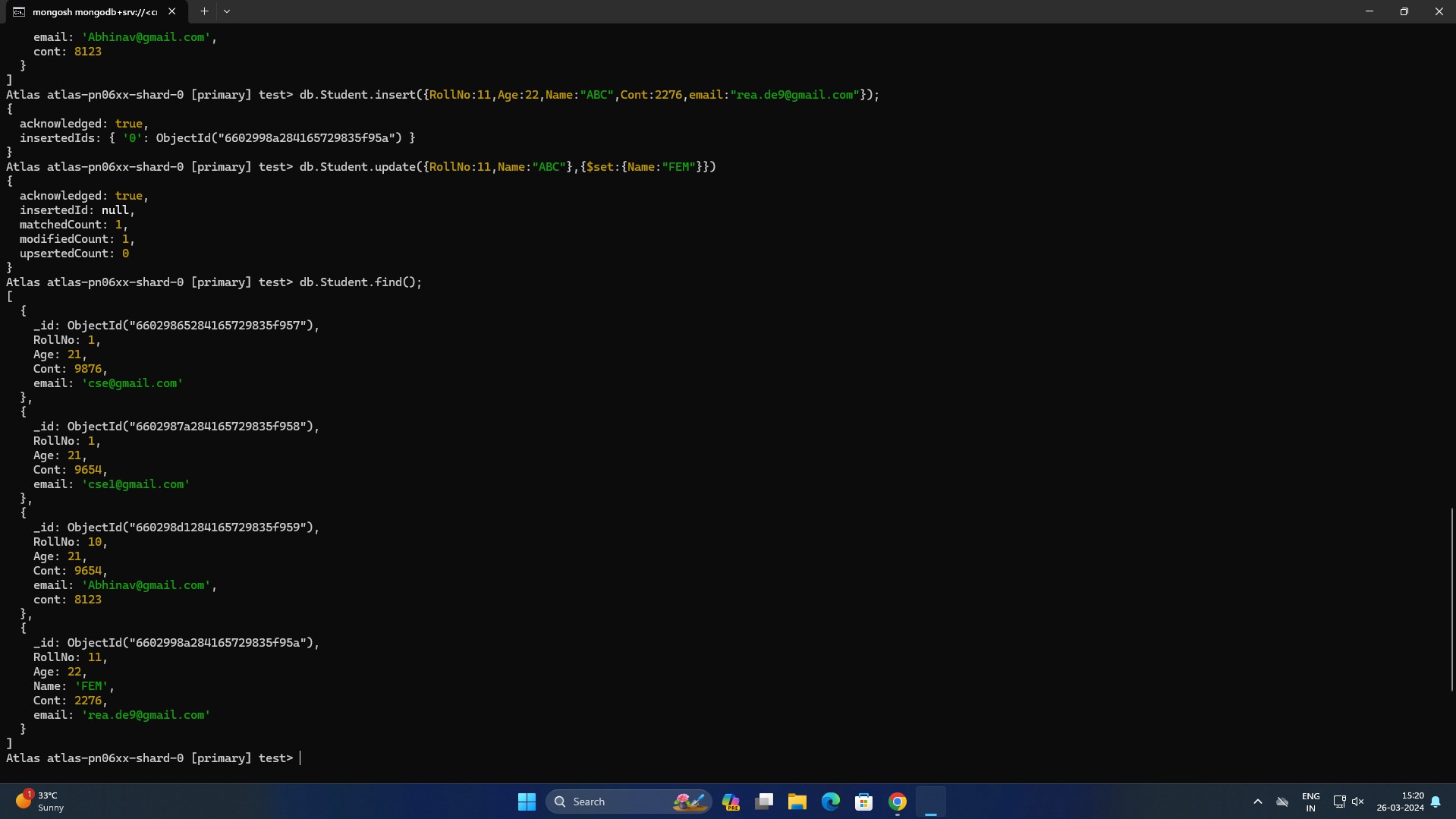
|  |  |  |
| --- | --- | --- |
| **Sl. No.** | **Experiment Title** | **Page No.** |
| 1. | MongoDB CRUD Operations | 4-9 |
| 2. | Cassandra Employee | 10-13 |
| 3. | Cassandra Library | 14-16 |
| 4. | Hadoop Installation | 17 |
| 5. | Hadoop Commands | 18-19 |
| 6. | Implement WordCount Program on Hadoop Framework | 20-25 |
| 7. | Implement Weather Data Program on Hadoop Framework | 26-33 |
| 8. | Implement Alphabetic Sort Program on Hadoop Framework | 34-40 |

**Course Outcome**

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

**MongoDB CRUD Operations**

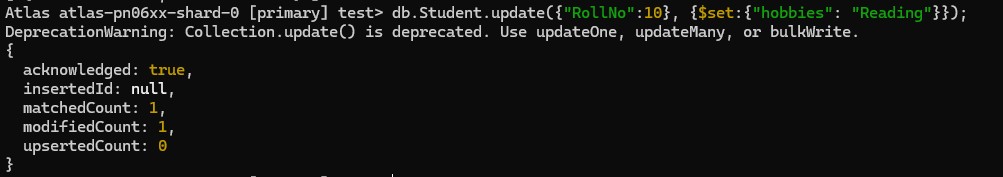
1. Create a database “Student” with the following attributes Rollno, Age, ContactNo, Email-Id. db.createCollection("Student");
2. Insert appropriate values [db.Student.insert({RollNo:1,Age:21,Cont:9876,email:"cse@gmail.com"});](mailto:cse@gmail.com) [db.Student.insert({RollNo:1,Age:21,Cont:9654,email:"cse1@gmail.com"});](mailto:cse1@gmail.com) db.Student.insert({RollNo:10,Age:21,Cont:9654}); db.Student.insert({RollNo:1[1,Age:22,Name:"ABC",Cont:2276,email:"rea.de9@gmail.com"});](mailto:rea.de9@gmail.com)
3. Write query to update Email-Id of a student with rollno 10. db.Student.update({RollNo:10},{$set:{email:"[Abhinav@gmail.com](mailto:Abhinav@gmail.com)"}})
4. Replace the student name from “ABC” to “FEM” of rollno 11. db.Student.update({RollNo:11,Name:"ABC"},{$set:{Name:"FEM"}})

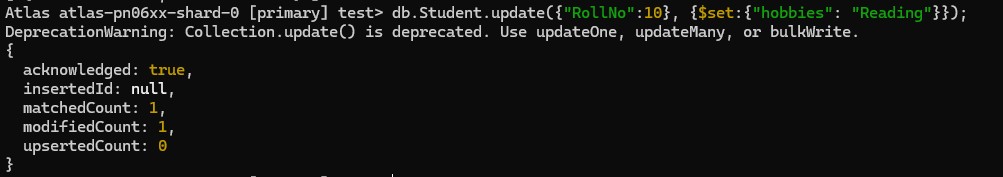


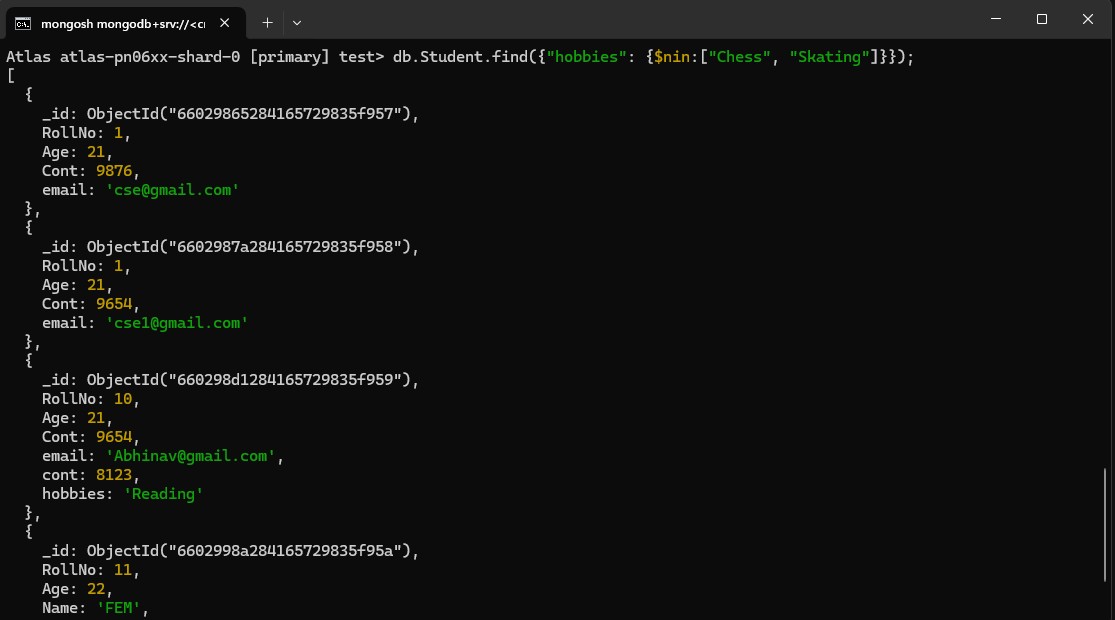
1. Display Student Name and grade(Add if grade is not present)where the \_id column is 1. db.Student.insert({\_id:1,Name:"David","Grade":"VII"});

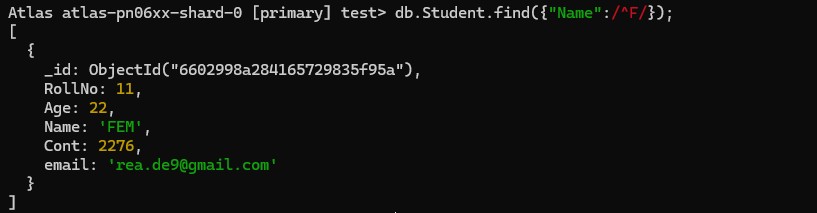
db.Student.find({\_id:1}, {Name:1, Grade:1, \_id:0});

1. Update to add hobbies

db.Student.update({"RollNo":10}, {$set:{"hobbies": "Reading"}});



1. Find documents where hobbies is set neither to Chess nor to Skating db.Student.find({"hobbies": {$nin:["Chess", "Skating"]}});
2. Find documents whose name begins with A db.Student.find({"Name":/^A/});

db.Student.find({"Name":/^F/});

1. Create a collection by name Customers with the following attributes. Cust\_id, Acc\_Bal,

Acc\_Type

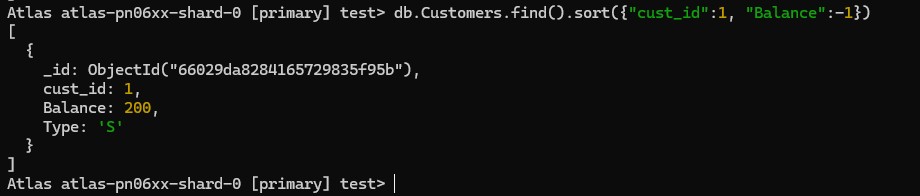
db.createCollection("Customers");

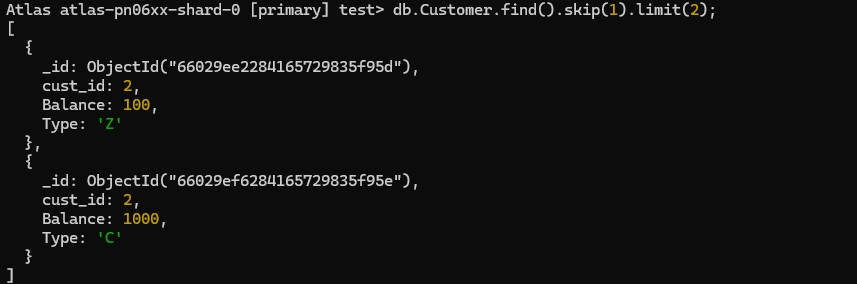
1. Insert at least 5 values into the table db.Customers.insert({cust\_id:1,Balance:200, Type:"S"}); db.Customer.insert({cust\_id:1, Balance:1000, Type:"Z"}); db.Customer.insert({cust\_id:2, Balance:100, Type:"Z"}); db.Customer.insert({cust\_id:2, Balance:1000, Type:"C"}); db.Customer.insert({cust\_id:3, Balance:500, Type:"Z"}); db.Customer.insert({cust\_id:2, Balance:50, Type:"S"});
2. Write a query to display those records whose total account balance is greater than 1200 of account type ‘Z’ for each customer\_id.

db.Customers.aggregate ( {$match:{Type:"Z"}}, {$group : { \_id : "$cust\_id", TotAccBal

:{$sum:"$Balance"} } }, {$match:{TotAccBal:{$gt:1200}}});

1. Determine Minimum and Maximum account balance for each customer\_id db.Customers.aggregate ( {$group : { \_id : "$cust\_id", minAccBal :{$min:"$Balance"}, maxAccBal :{$max:"$Balance"} }});
2. Sort the documents based on Customer ID in ascending order and Account Balance in descending order

db.Customers.find().sort({"cust\_id":1, "Balance":-1})

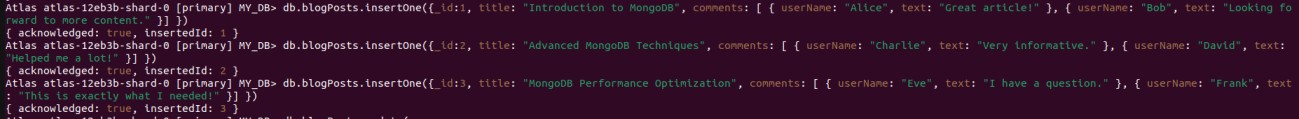
1. Display only 2 nd and 3 rd records from the collection db.Customer.find().skip(1).limit(2);

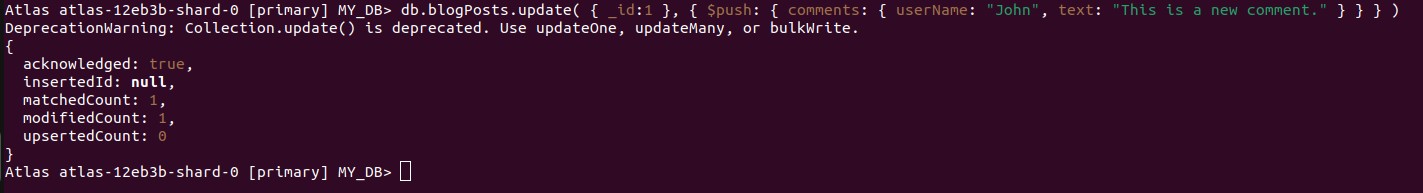
Create a collection by the name blogPosts and it has 3 fields id, title and comments.

In the collection the comments field is an array which consists of user details. Each collection consists of two user details inside the comments array- user name and text

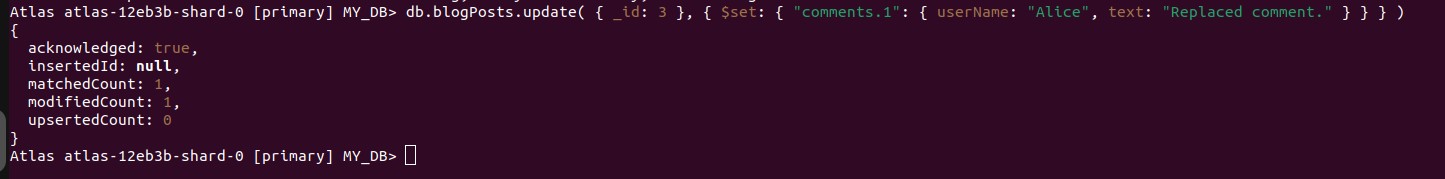
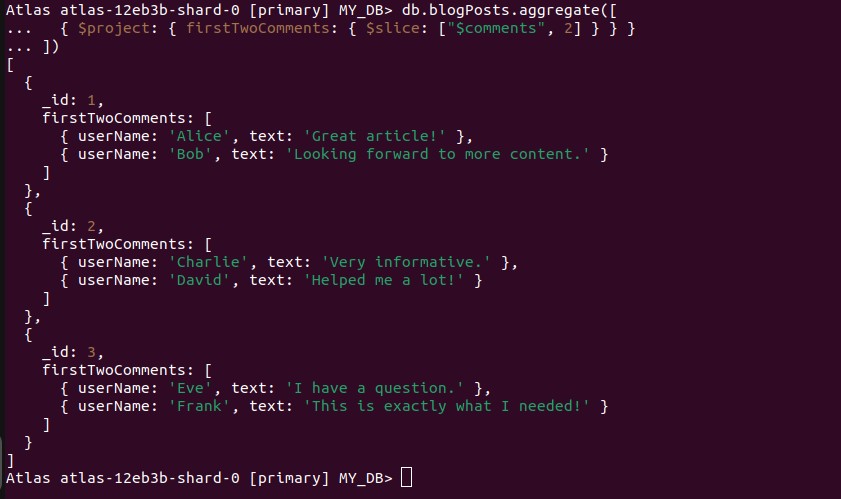
Demonstrate the following

1. Adding an element into array
2. Display second element
3. Display size of the array
4. Display first two elements of the array
5. Update the document with id 4 and replace the element present in 1st index position of the array with another array









**Cassandra Employee**

1.Create a keyspace by name Employee

CREATE KEYSPACE Employee WITH REPLICATION = {'class':'SimpleStrategy', 'replication\_factor':1};

use Employee;

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

CREATE TABLE Employee\_Info(Emp\_id int PRIMARY KEY,Emp\_Name text, Designation text, Date\_of\_Joining timestamp, Salary double, Dept\_Name text);

1. Insert the values into the table in batch Begin batch

insert into Employee\_Info(Emp\_id,Emp\_Name,Designation, Date\_of\_Joining, Salary,Dept\_Name)

VALUES (111,'John','Manager','2010-02-27',80000.0,'IT')

insert into Employee\_Info(Emp\_id,Emp\_Name,Designation, Date\_of\_Joining, Salary,Dept\_Name)

VALUES (121,'James','Developer','2019-06-27',60000.0,'IT')

insert into Employee\_Info(Emp\_id,Emp\_Name,Designation, Date\_of\_Joining, Salary,Dept\_Name)

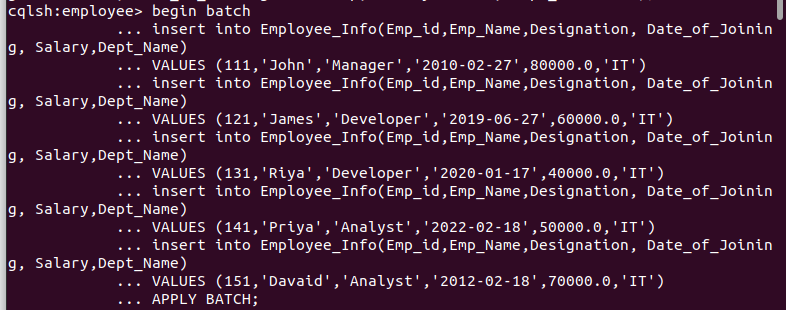
VALUES (131,'Riya','Developer','2020-01-17',40000.0,'IT')

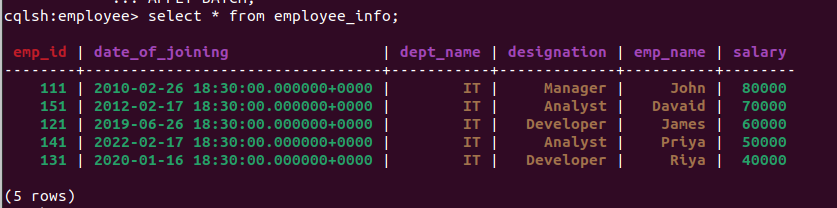
insert into Employee\_Info(Emp\_id,Emp\_Name,Designation, Date\_of\_Joining, Salary,Dept\_Name)

VALUES (141,'Priya','Analyst','2022-02-18',50000.0,'IT')

insert into Employee\_Info(Emp\_id,Emp\_Name,Designation, Date\_of\_Joining, Salary,Dept\_Name)

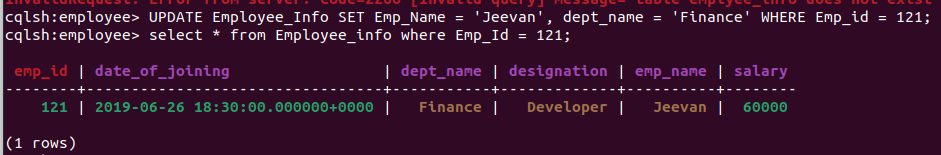
VALUES (151,'Davaid','Analyst','2012-02-18',70000.0,'IT') APPLY BATCH

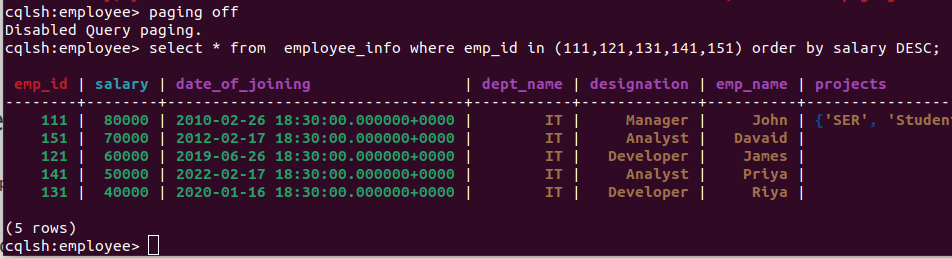


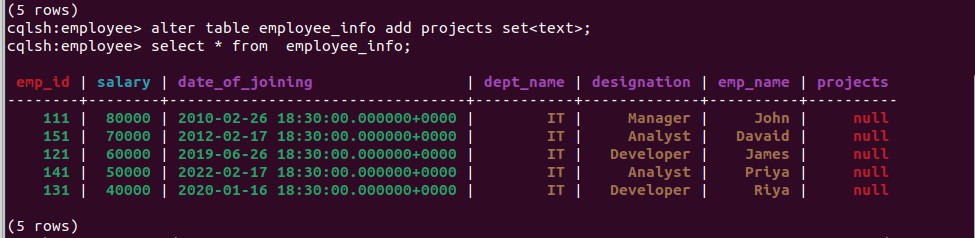


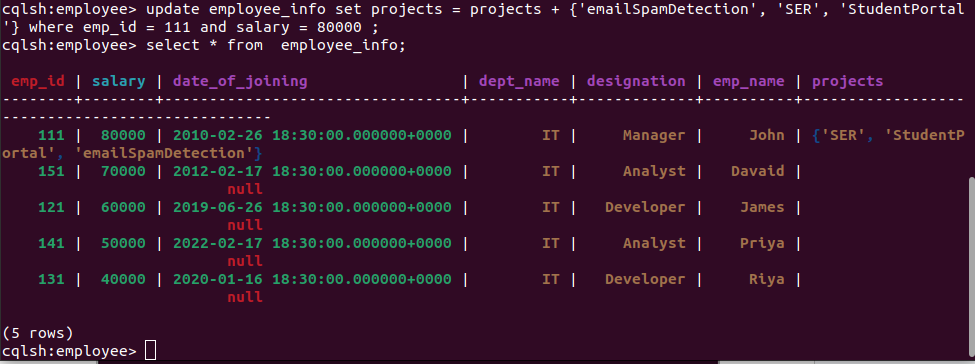
1. Update Employee name and Department of Emp-Id 121

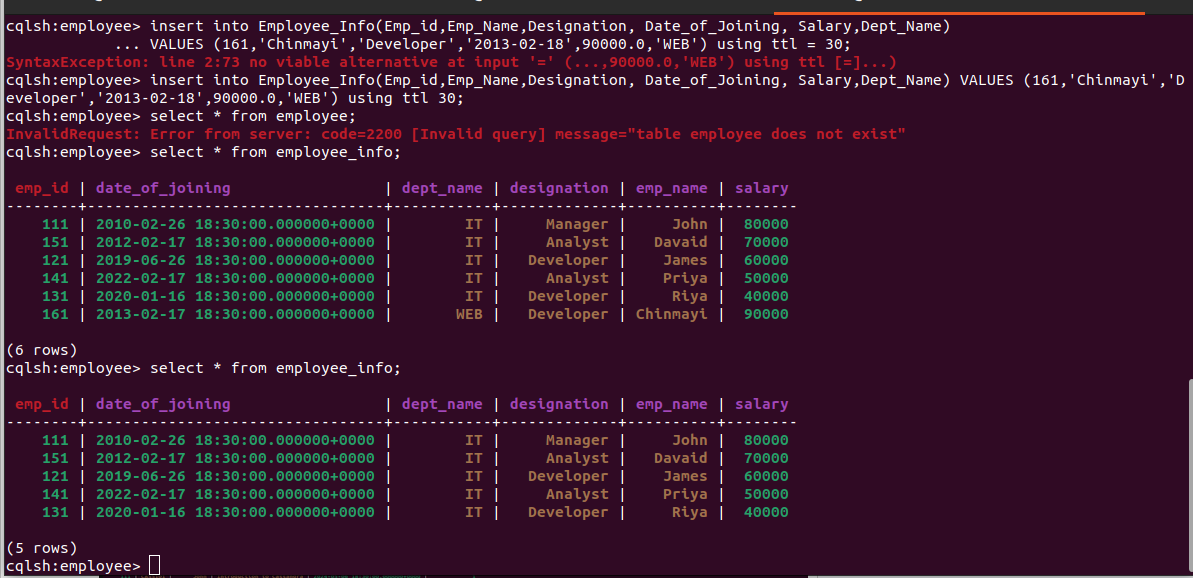
UPDATE Employee\_Info SET Emp\_Name = 'Jeevan', Department = 'Finance' WHERE Emp\_id

= 121;

1. Sort the details of Employee records based on salary
2. Alter the schema of the table Employee Info to add a column Projects which stores a set of Projects done by the corresponding Employee.



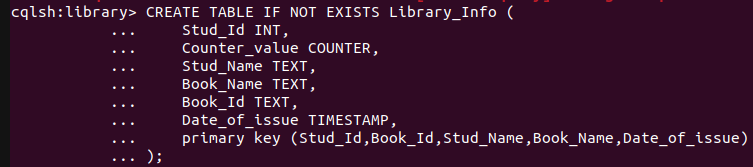
1. Update the altered table to add project names.
2. Create a TTL of 15 seconds to display the values of Employees



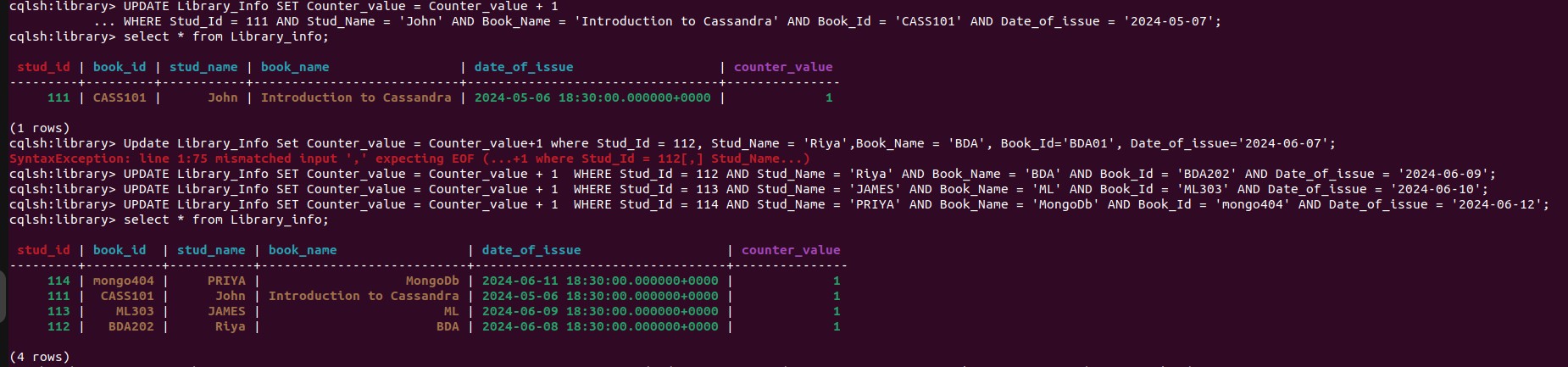
**Cassandra Library**

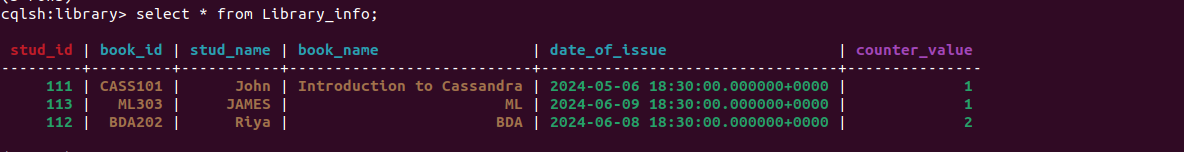
Perform the following DB operations using Cassandra. 1.Create a keyspace by name Library

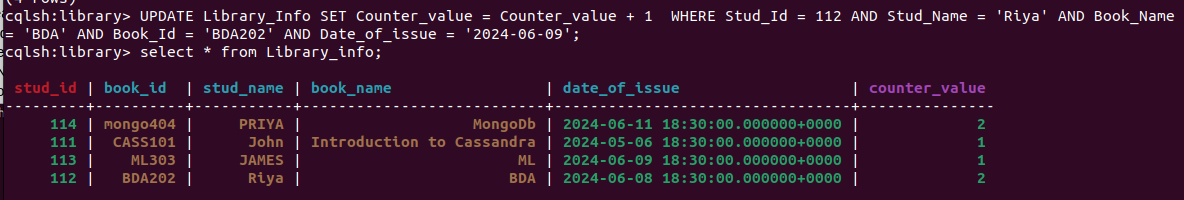
1. Create a column family by name Library-Info with attributes Stud\_Id Primary Key,

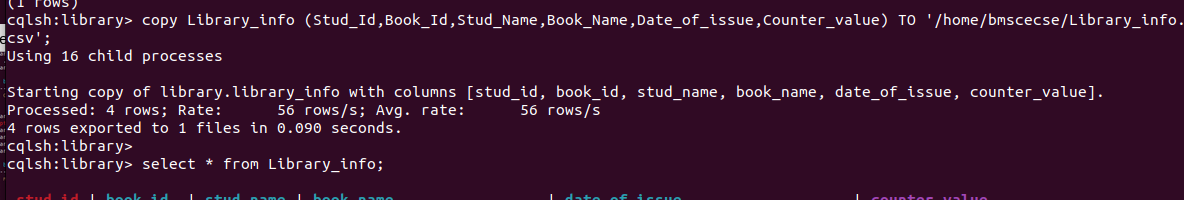
Counter\_value of type Counter, Stud\_Name, Book-Name, Book-Id, Date\_of\_issue

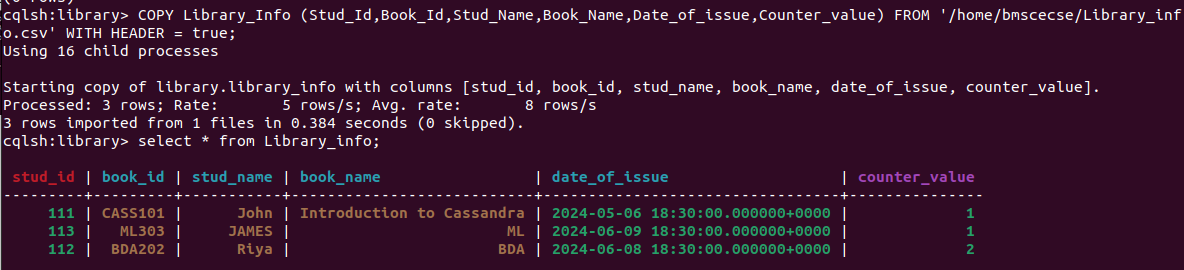
1. Insert the values into the table in batch

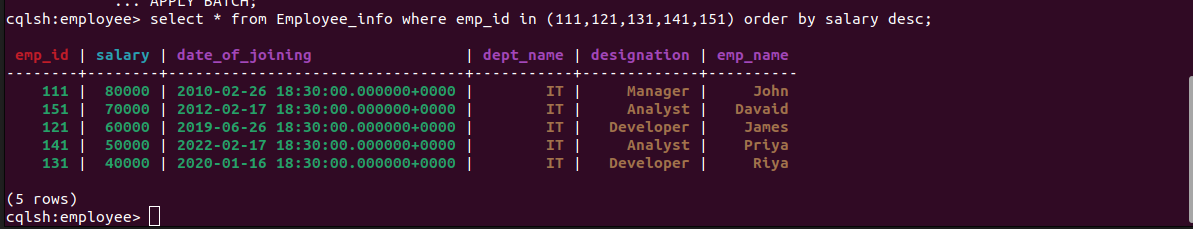


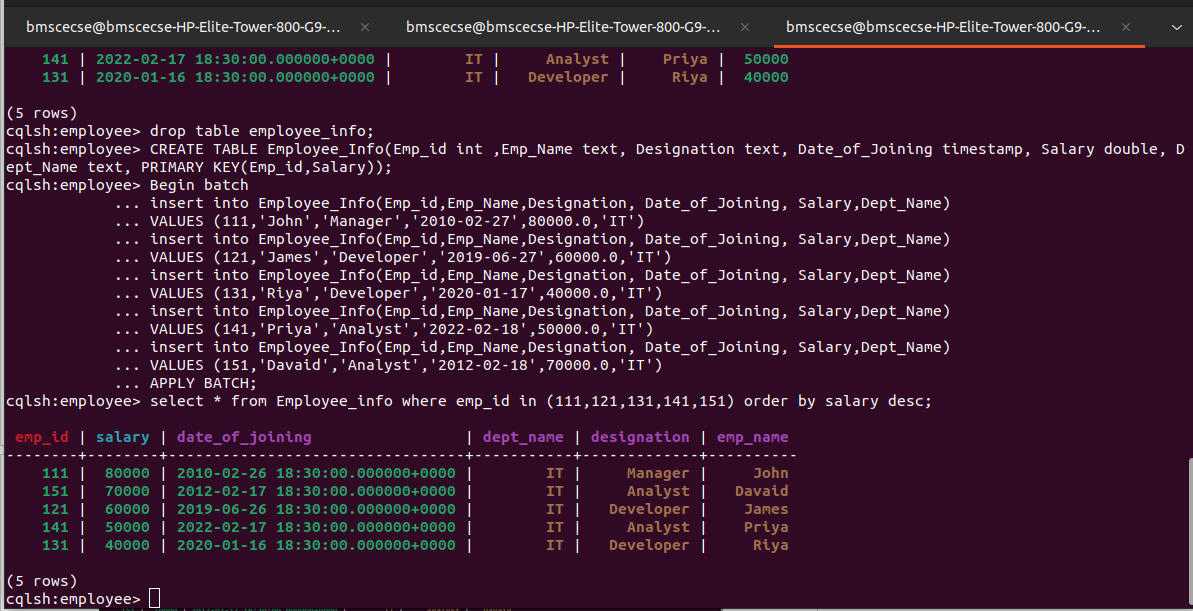
1. Display the details of the table created and increase the value of the counter
2. Write a query to show that a student with id 112 has taken a book “BDA” 2 time



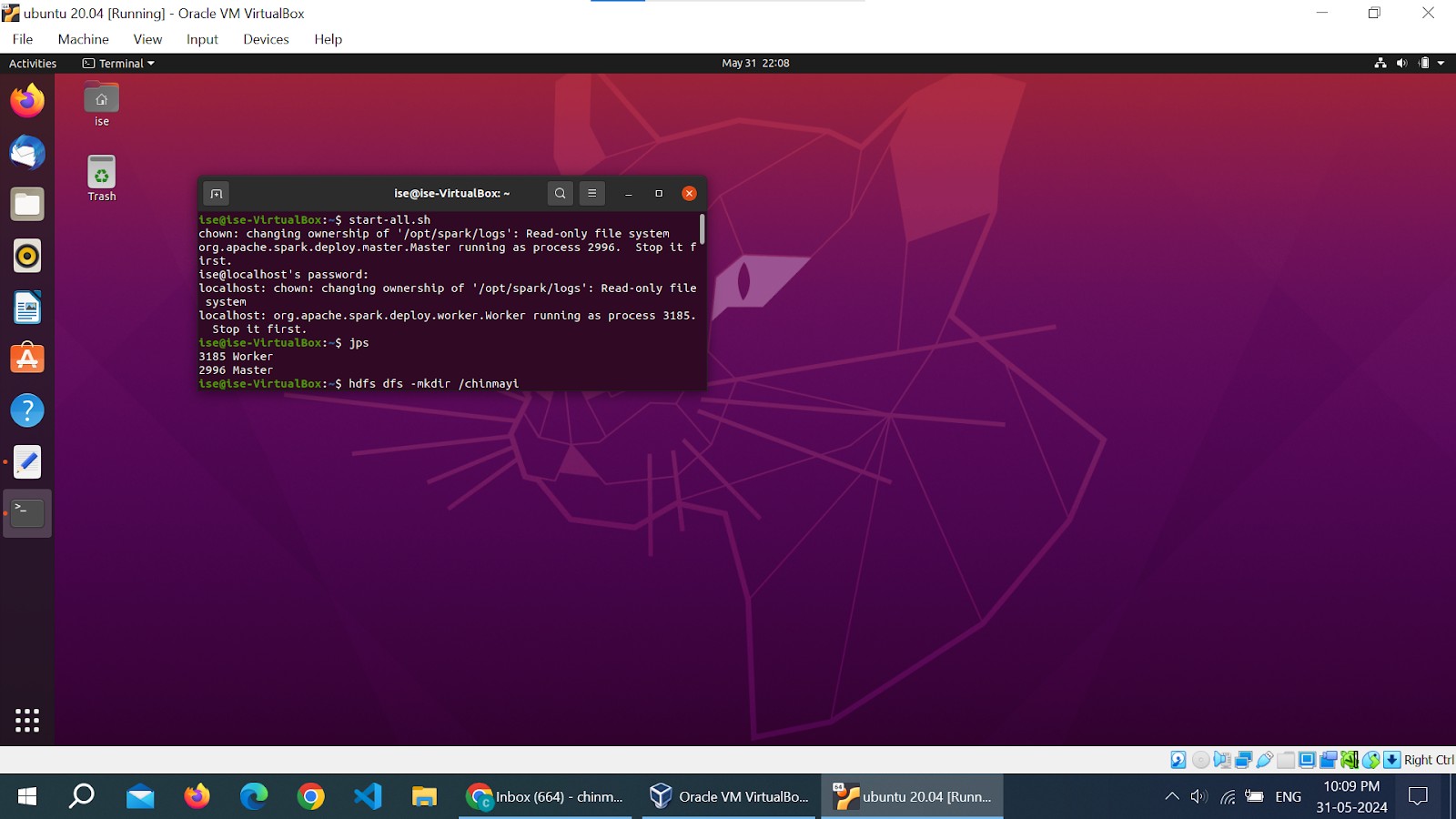
1. Export the created column to a csv file
2. Import a given csv dataset from local file system into Cassandra column family





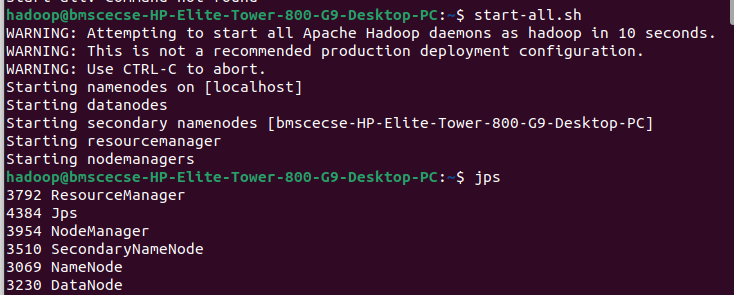


**Hadoop Installation (ScreenShot)**



**Hadoop Commands**

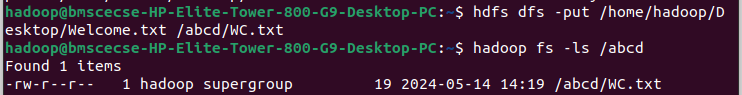
(Minimum 10 commands to be executed).



* 1. mkdir: Hadoop HDFS mkdir Command Usage

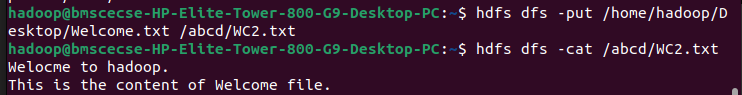


* 1. lsHadoop HDFS ls Command Usage
  2. putHadoop HDFS put Command Usage



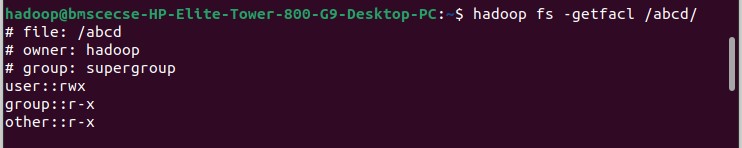
* 1. copyFromLocal

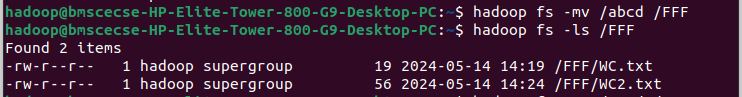
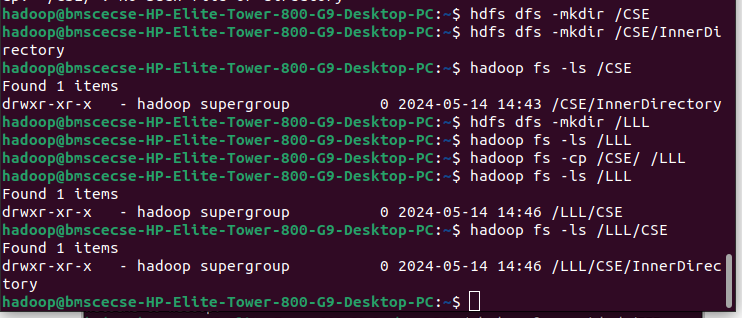
Hadoop HDFS copyFromLocal Command



* 1. getHadoop HDFS get Command Usage i.Hadoop HDFS get Command Example

1. Hadoop HDFS get Command Example
2. Hadoop HDFS get Command



* 1. copyToLocalHadoop
  2. Cat Hadoop HDFS cat Command Usage
  3. mvHadoop HDFS mv Command Usage
  4. Cp Hadoop HDFS

**Implement WordCount Program on Hadoop framework**

**Mapper Code:**

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> {

public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output, Reporter rep) throws IOException

{

String line = value.toString(); for (String word : line.split(" "))

{

if (word.length() > 0)

{

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

} } } }

**Reducer Code:**

// 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

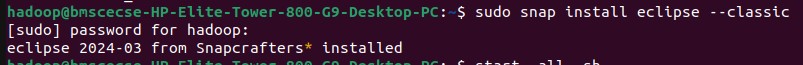
public static void main(String args[]) throws Exception

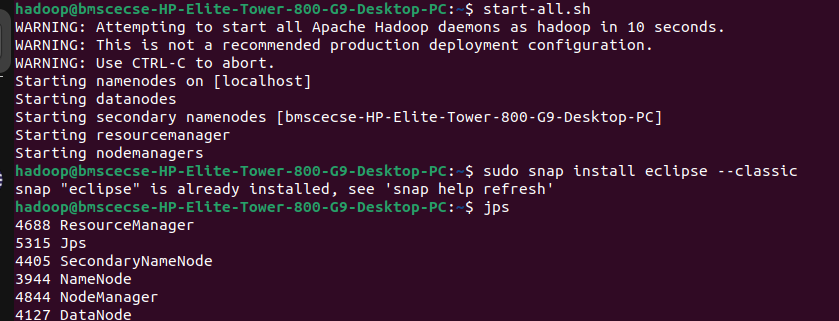
{

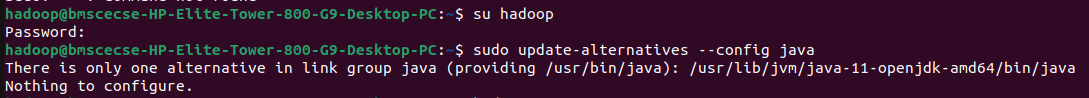
int exitCode = ToolRunner.run(new WCDriver(), args); System.out.println(exitCode);

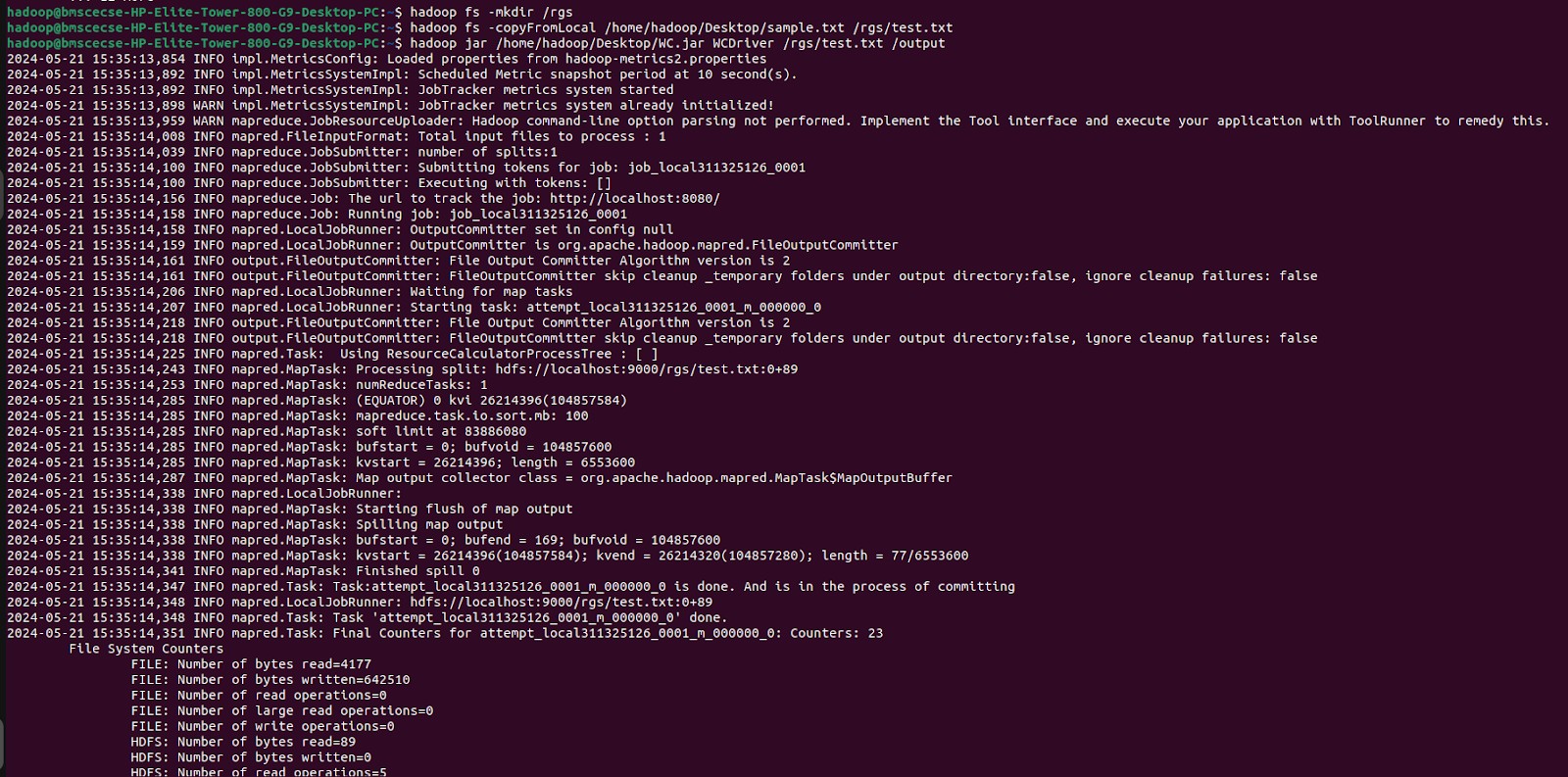
}

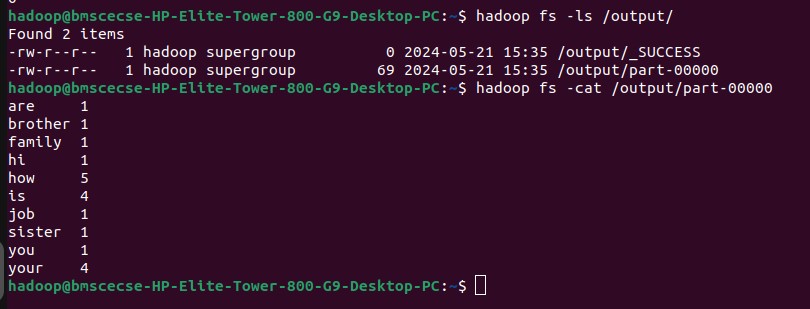
}











**Implement Weather Data Program on Hadoop framework**

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

**Create a Map Reduce program to**

1. **Find average temperature for each year from 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<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 {

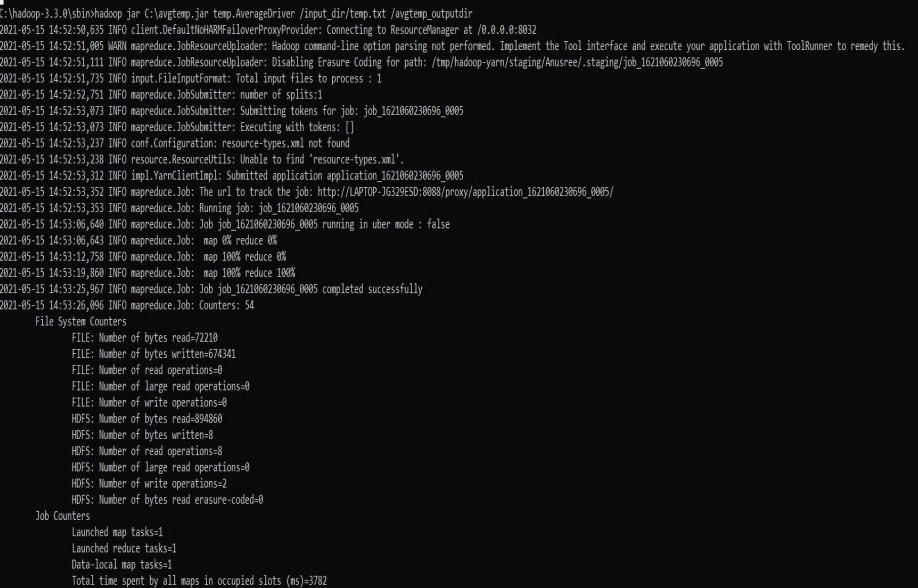
int max\_temp = 0; int count = 0;

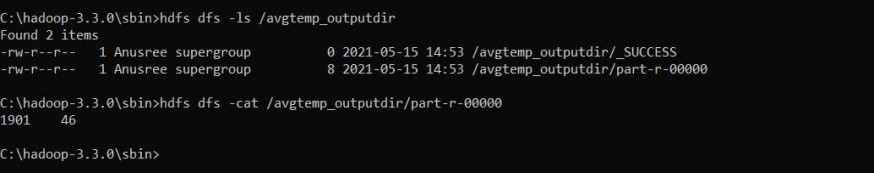
for (IntWritable value : values) { max\_temp += value.get(); count++;

}

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

}}





1. **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<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++;

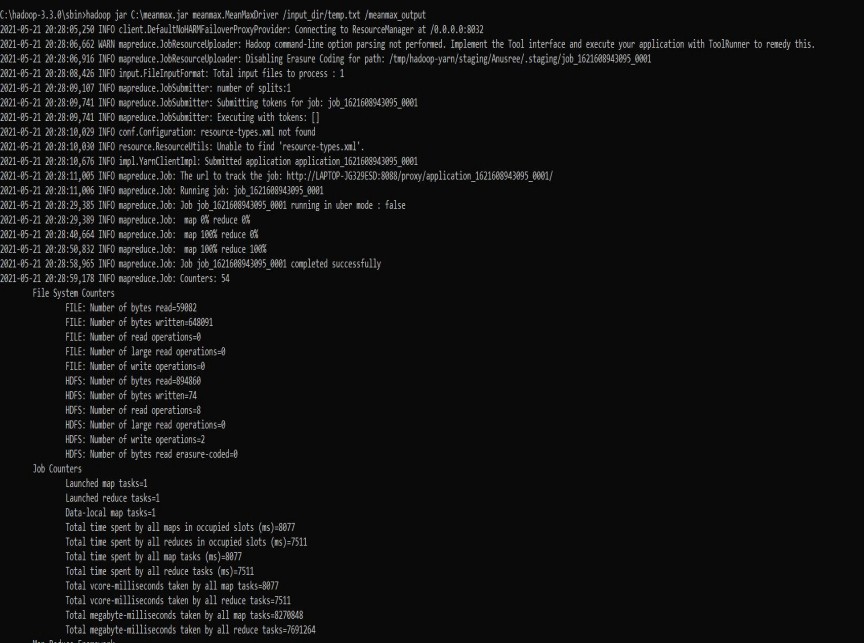
}

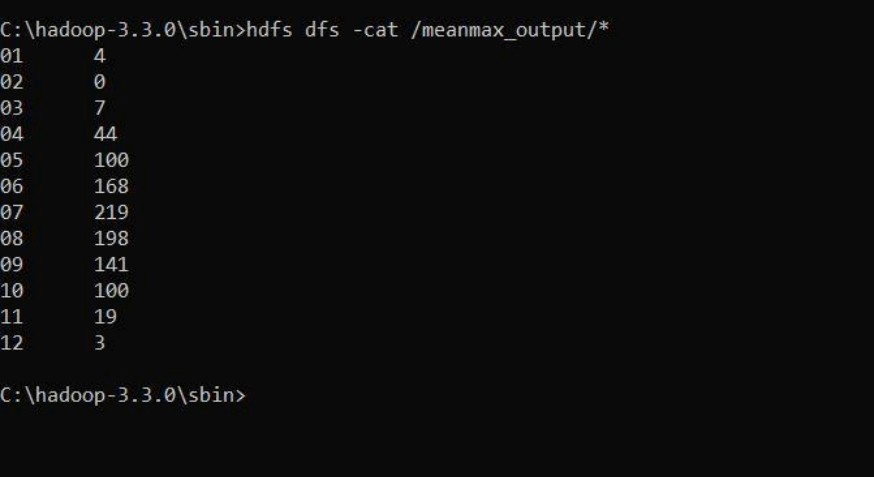
}

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

}

}





**Implement Alphabetic Sort Program on Hadoop framework**

**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.**

**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 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);

}

}

}

**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));

}

}

}

