**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

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



**LAB REPORT**

**on**

**COURSE TITLE**

***Submitted by***

**S Swaroop Bharadwaj**

***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-2023 to July-2023**

**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 **S Swaroop Bharadwaj (1BM20CS136),** who is bonafide student of **B. M. S. College of Engineering.** It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2022. The Lab report has been approved as it satisfies the academic requirements in respect of a **Big Data Analytics - (20CS6PEBDA)** work prescribed for the said degree.

Vikranth BM **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** |
| **1** | **Cassandra Lab Program1: - Employee Database** |
| **2** | **Cassandra Lab Program1: - Library Database** |
| **3** | **MongoDB- CRUD Demonstration** |
| **4** | **Hadoop installation** |
| **5** | **Hadoop Commands** |
| **6** | **Hadoop Program: Average Temperature** |
| **7** | **Hadoop Program: Word Count** |
| **8** | **Hadoop program: Join operation** |
| **9** | **Scal Program** |
| **10** | **Scala Program: Word Count** |

**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 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**

1. **Insert the values into the table in batch**
2. **Update Employee name and Department of Emp-Id 121**
3. **Sort the details of Employee records based on salary**
4. **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.**

cqlsh:employee> CREATE KEYSPACE employee WITH REPLICATION={ 'class'

: 'SimpleStrategy', 'replication\_factor' : 1}; cqlsh:employee> USE employee;

cqlsh:employee> create table employee\_info(emp\_id int PRIMARY KEY, emp\_name text, ... designation text, date\_of\_joining timestamp, salary double PRIMARY KEY, dept\_name text);

cqlsh:employee> CREATE TABLE employee\_info(emp\_id int, emp\_name text, designation text, date\_of\_joining timestamp, salary double, dept\_name text, PRIMARY KEY(emp\_id, salary));

cqlsh:employee> BEGIN BATCH INSERT INTO

...

employee\_info(emp\_id,emp\_name,designation,date\_of\_joining,sal ary,dept\_name) ...

VALUES(100,'John','MANAGER','2021-09-11',30000,'TESTING');

... INSERT INTO

...

employee\_info(emp\_id,emp\_name,designation,date\_of\_joining,sal ary,dept\_name) ...

VALUES(111,'Tom','ASSOCIATE','2021-06-22',25000,'DEVELOPING');

... INSERT INTO

...

employee\_info(emp\_id,emp\_name,designation,date\_of\_joining,sal ary,dept\_name) ...

VALUES(121,'Elsa','MANAGER','2021-03-30',35000,'HR');

... INSERT INTO

...

employee\_info(emp\_id,emp\_name,designation,date\_of\_joining,sal ary,dept\_name) ...

VALUES(115,'Chris','ASSISTANT','2021-12-30',20000,'DEVELOPING');

... INSERT INTO

...

employee\_info(emp\_id,emp\_name,designation,date\_of\_joining,sal ary,dept\_name) ...

VALUES(105,'Sarah','ASSOCIATE','2021-06-25',25000,'TESTING');

... APPLY BATCH;

cqlsh:employee> SELECT \* FROM employee\_info

... ;

cqlsh:employee> UPDATE employee\_info SET emp\_name = 'Jessica', dept\_name = 'DEVELOPING' WHERE emp\_id = 121;

cqlsh:employee> UPDATE employee\_info SET emp\_name = 'Jessica', dept\_name = 'DEVELOPING' WHERE emp\_id = 121 AND salary = 35000;

cqlsh:employee> SELECT \* FROM employee\_info ;

MANAGER | John

cqlsh:employee> SELECT \* FROM employee\_info WHERE emp\_id in (105, 111, 121, 115, 100) order by salary; cqlsh:employee> paging off

Disabled Query paging.

cqlsh:employee> SELECT \* FROM employee\_info WHERE emp\_id in (105, 111, 121, 115, 100) order by salary;

cqlsh:employee> ALTER TABLE employee\_info ADD projects text;

cqlsh:employee> UPDATE employee\_info SET projects = 'Chat App' WHERE emp\_id = 111; cqlsh:employee> UPDATE employee\_info SET projects = 'Chat App' WHERE emp\_id = 111 and salary = 25000;

cqlsh:employee> UPDATE employee\_info SET projects = 'Discord Bot' WHERE emp\_id

= 115 and salary = 20000;

cqlsh:employee> UPDATE employee\_info SET projects = 'Campus Portal' WHERE emp\_id

= 105 and salary = 25000;

cqlsh:employee> UPDATE employee\_info SET projects = 'YouTube Downloader' WHERE emp\_id = 100 and salary = 30000;

cqlsh:employee> UPDATE employee\_info SET projects = 'Library Management System ' WHERE emp\_id = 121 and salary = 35000;

cqlsh:employee> SELECT \* FROM employee\_infor

... ;

cqlsh:employee> SELECT \* FROM employee\_info ; cqlsh:employee> INSERT INTO

...

employee\_info(emp\_id,emp\_name,designation,date\_of\_joining,sal ary,dept\_name) ...

... ;

cqlsh:employee> INSERT INTO

...

employee\_info(emp\_id,emp\_name,designation,date\_of\_joining,sal ary,dept\_name) ...

VALUES(110,'SAM','ASSOCIATE','2021-01-11',28000,'TESTING') USING TTL 15;

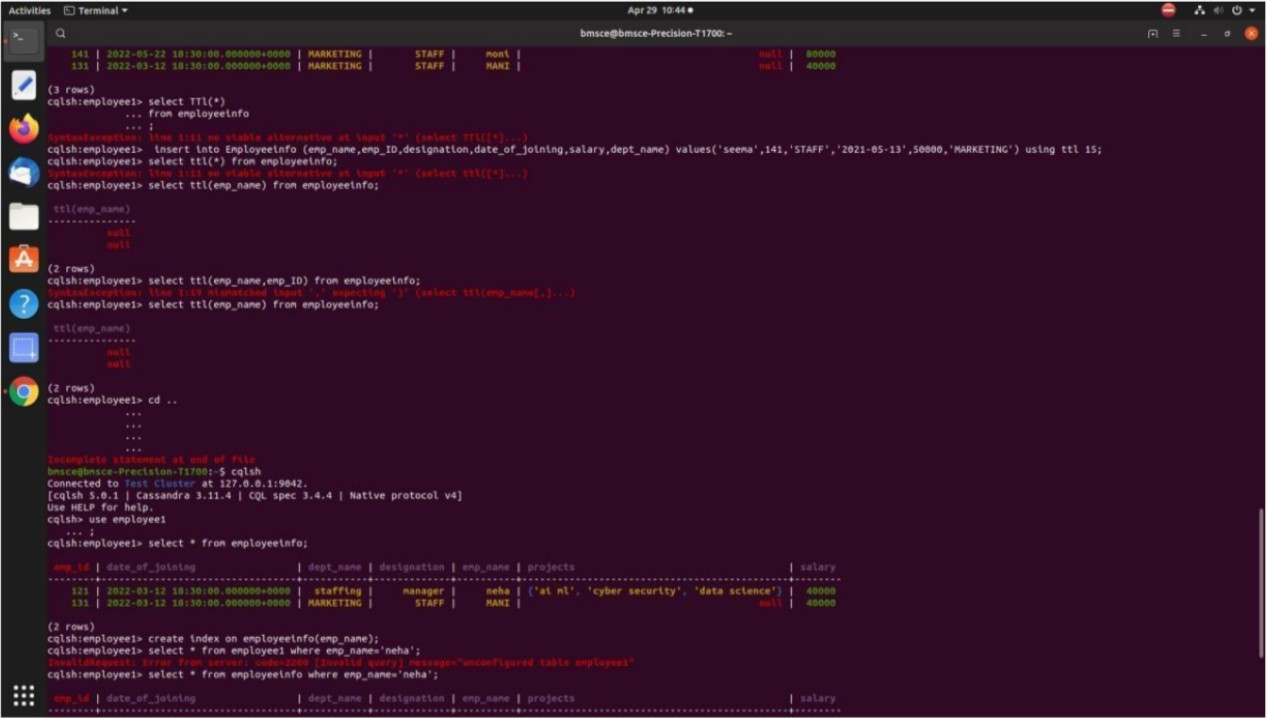
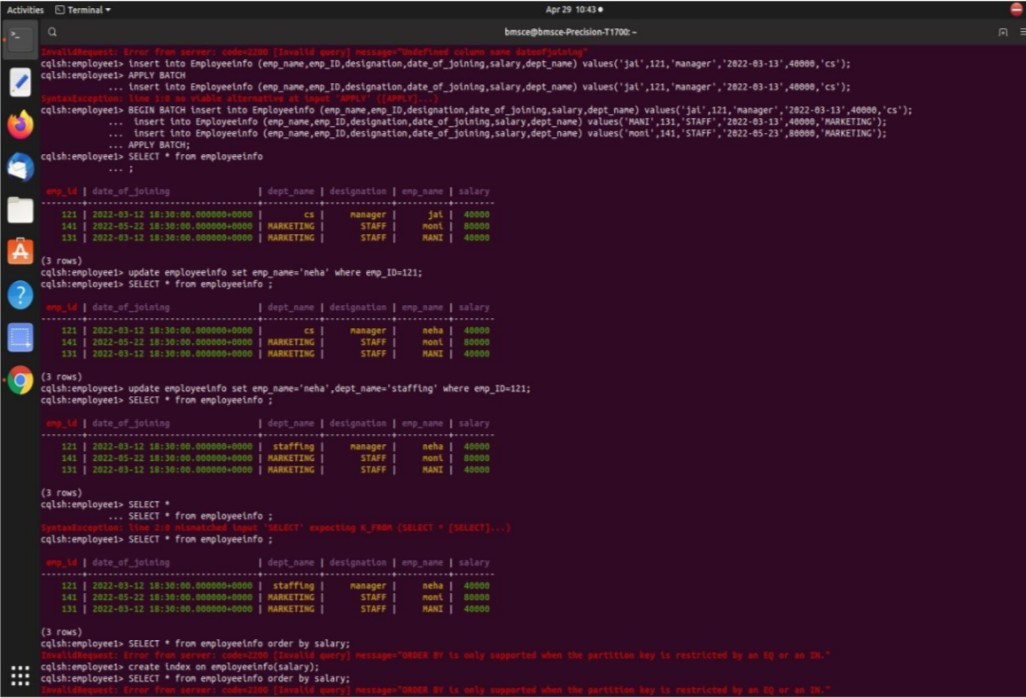
cqlsh:employee> SELECT TTL(emp\_name) from employee\_info

WHERE emp\_id = 110; ttl(emp\_name)

3

cqlsh:employee> SELECT \* FROM employee\_info;

**Output:**



# 2.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** cqlsh:library> CREATE KEYSPACE library WITH replication = {'class': 'SimpleStrategy','replication\_factor':1}; cqlsh:library> USE library ;

cqlsh:library> CREATE TABLE Library\_info(stud\_id int, stud\_name text, book\_name text, book\_id text, date\_of\_issue timestamp, counter\_value counter, PRIMARY KEY(stud\_id,stud\_name, book\_name, book\_id, date\_of\_issue));

cqlsh:library> BEGIN COUNTER BATCH

... UPDATE library\_info set counter\_value +=1 where stud\_id = 111 and stud\_name = 'Manoj' and book\_name = 'Operations Research' and book\_id = '56TXT' and date\_of\_issue = '2021-09-12';

... UPDATE library\_info set counter\_value +=1 where stud\_id = 112 and stud\_name = 'Kamal' and book\_name = 'Engineering Mathematics-3' and book\_id = '5ERW4' and date\_of\_issue = '2021-04-10';

... UPDATE library\_info set counter\_value +=1 where stud\_id = 113 and stud\_name = 'Mahesh' and book\_name = 'Robinson Crusoe' and book\_id = '34EDC' and date\_of\_issue = '2021-02-01';

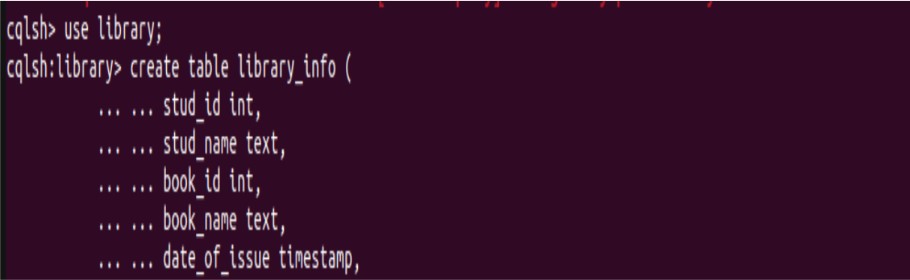
... UPDATE library\_info set counter\_value +=1 where stud\_id = 114 and stud\_name = 'Raj' and book\_name = 'Engineering Drawing' and book\_id = '123ER' and date\_of\_issue = '2021-04-03';

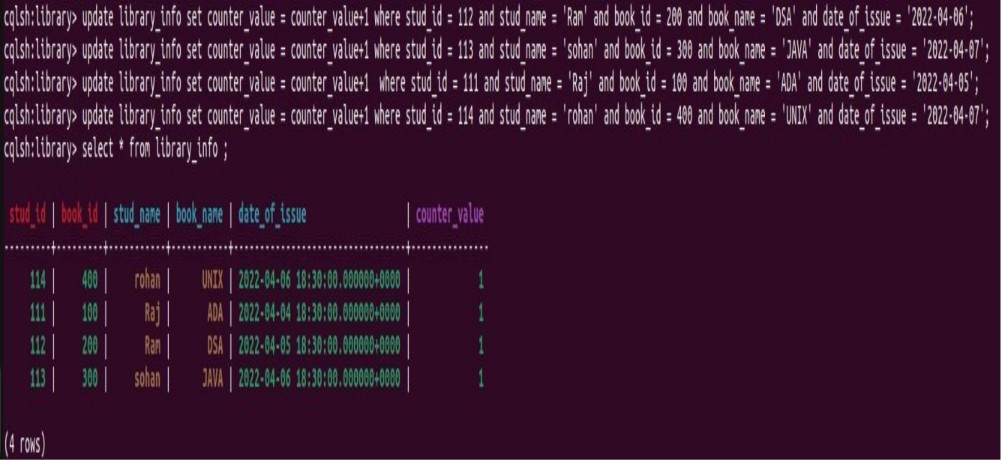
... APPLY BATCH;

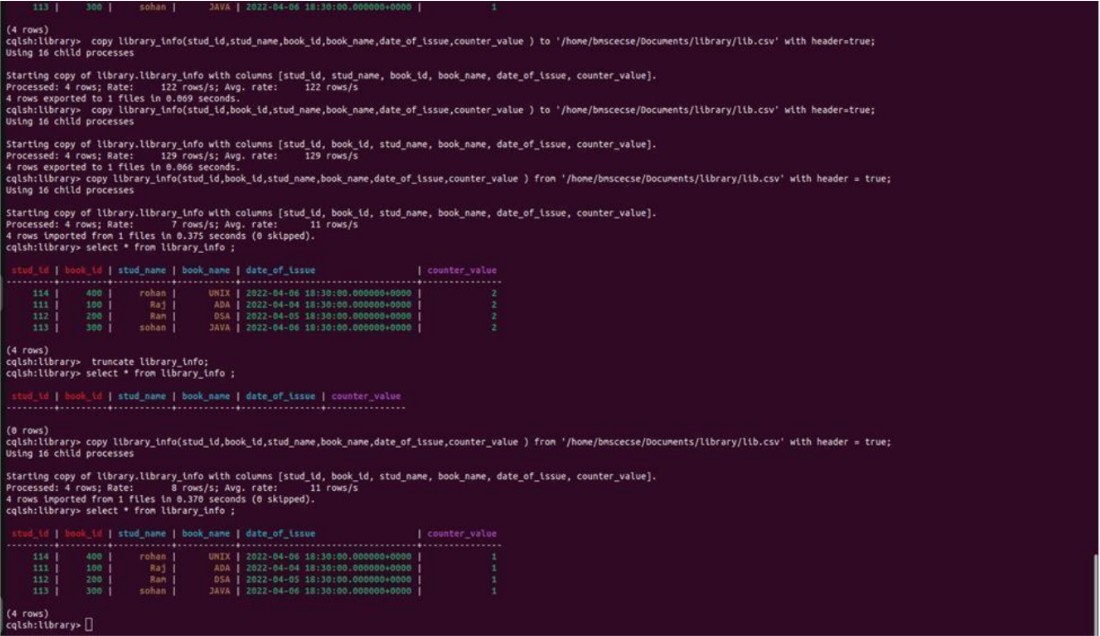
cqlsh:library> SELECT \* FROM library\_info ;

cqlsh:library> UPDATE library\_info set counter\_value += 1 where stud\_id = 112 and stud\_name = 'Kamal' and book\_name = 'Engineering Mathematics-3' and book\_id = '5ERW4' and date\_of\_issue = '2021-04-09';

cqlsh:library> SELECT \* FROM library\_info ;







1. **MongoDB- CRUD Demonstration** bmsce@bmsce-Precision-T1700:~$ mongo MongoDB shell version v3.6.8

connecting to: mongodb://127.0.0.1:27017 Implicit session: session { "id" :

UUID("d66acdb3-8482-417d-8b75-d65dae4b53ee") } MongoDB server version: 3.6.8

* use Student switched to db Student
* db.createCollection("student");

{ "ok" : 1 }

>

db.Student.insert({\_id:1,StudName:"Megha",Grade:"vii",Hobbies:"Int ernetSurfi ng"}); WriteResult({ "nInserted" : 1 })

>

db.Student.update({\_id:3,StudName:"Ayan",Grade:"vii"},{$set:{Hobbies:"skating"}

},{upsert:t rue}); WriteResult({ "nMatched" : 0, "nUpserted" : 1, "nModified" : 0, "\_id" : 3 })

* db.Student.find({StudName:"Ayan"}

{ "\_id" : 3, "Grade" : "vii", "StudName" : "Ayan", "Hobbies" : "skating" }

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

{ "StudName" : "Megha", "Grade" : "vii" }

{ "Grade" : "vii", "StudName" : "Ayan" }

* + db.Student.find({Grade:{$eq:'vii'}}).pretty();

{

"\_id" : 1,

"StudName" : "Megha",

"Grade" : "vii",

"Hobbies" : "InternetSurfing"

}

{ "\_id" : 3, "Grade" : "vii", "StudName" : "Ayan", "Hobbies" : "skating" } > db.Student.find({Grade:{$eq:'vii'}});

{ "\_id" : 1, "StudName" : "Megha", "Grade" : "vii", "Hobbies" :

"InternetSurfing" } { "\_id" : 3, "Grade" : "vii", "StudName" : "Ayan", "Hobbies" : "skating" } > db.Student.find({Grade:{$eq:'vii'}}).pretty();

{

"\_id" : 1,

"StudName" : "Megha",

"Grade" : "vii",

"Hobbies" : "InternetSurfing"

}

{ "\_id" : 3, "Grade" : "vii", "StudName" : "Ayan", "Hobbies" : "skating" } > db.Student.find({Hobbies:{$in:['Chess','Skating']}}).pr etty(); > db.Student.find({Hobbies:{$in:['Skating']}}).pretty();

* db.Student.find({Hobbies:{$in:['skating']}}).pretty();

{ "\_id" : 3, "Grade" : "vii", "StudName" : "Ayan", "Hobbies" : "skating" }

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

{

"\_id" : 1,

"StudName" : "Megha",

"Grade" : "vii",

"Hobbies" : "InternetSurfing"

}

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

{

"\_id" : 1,

"StudName" : "Megha",

"Grade" : "vii",

"Hobbies" : "InternetSurfing"

}

* db.Student.c ount(); 2
* db.Student.find().sort({StudName:-1}).pretty();

{

"\_id" : 1,

"StudName" : "Megha",

"Grade" : "vii",

"Hobbies" : "InternetSurfing"

}

{ "\_id" : 3, "Grade" : "vii", "StudName" : "Ayan", "Hobbies" : "skating" } > db.Student.save({StudName:"Vamsi",Greade

:"vi"}) WriteResult({ "nInserted" : 1 })

>

db.Students.update({\_id:4},{$set:{Location:"N etwork"}}) WriteResult({ "nMatched" : 0,

"nUpserted" : 0, "nModified" : 0 })

>

db.Students.update({\_id:4},{$unset:{Location: "Network" }}) WriteResult({ "nMatched" : 0, "nUpserted" : 0, "nModified" : 0 })

>

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

"Megha", "Grade" : "vii" }

* db.Student.find({Grade:{$ne:'VII'}}).pretty();

{

"\_id" : 1,

"StudName" : "Megha",

"Grade" : "vii",

"Hobbies" : "InternetSurfing"

}

{ "\_id" : 3, "Grade" : "vii", "StudName" : "Ayan", "Hobbies" : "skating" } {

"\_id" : ObjectId("6253f413e88b8c9e787b194e"), "StudName" : "Vamsi",

"Grade" : "vi"

}

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

>

db.Students.update({\_id:3},{$set:{Location:null

}}) WriteResult({ "nMatched" : 0, "nUpserted" :

0, "nModified" : 0 }) > db.Students.c ount() 0

* db.Students.count({Grade:

"VII"}) 0

* db.Student.find({Grade:"VII"}).limit(3).pretty();
* db.Student.update({\_id:3},{$set:{Location:null}})

WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

* db.Student.count({Grade:"VII"}) 0
* db.Students.count({Grade: "vii"}) 0
* db.Student.c ount() 3
* db.Student.count({Grade: "vii"}) 2

>

db.Student.find({Grade:"vii"}). limit(3).pretty(); {

"\_id" : 1,

"StudName" : "Megha",

"Grade" : "vii",

"Hobbies" : "InternetSurfing"

}

{

"\_id" : 3, "Grade" : "vii",

"StudName" : "Ayan",

"Hobbies" : "skating", "Location" : null

}

>

db.Student.find().sort({StudN ame:1}).pretty(); {

"\_id" : 3,

"Grade" :

"vii",

"StudName" : "Ayan",

"Hobbies" : "skating", "Location" : null

}

{

"\_id" : 1,

"StudName" : "Megha",

"Grade" : "vii",

"Hobbies" : "InternetSurfing"

}

{

"\_id" : ObjectId("6253f413e88b8c9e787b194e"), "StudName" : "Vamsi",

"Grade" : "vi"

}

* db.Student.find().skip(2).pretty()

{

"\_id" : ObjectId("6253f413e88b8c9e787b194e"),

"StudName" : "Vamsi",

"Grade" : "vi"

}

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

2022-04-11T15:05:51.894+0530 E QUERY [thread1] SyntaxError: missing ] after element list @(shell):1:57 > db.food.insert({\_id:1,fruits:['grapes','mango','ap

ple']}) WriteResult({ "nInserted" : 1 })

* db.food.insert({\_id:2,fruits:['grapes','mango','che rry']}) WriteResult({ "nInserted" : 1 })
* db.food.insert({\_id:3,fruits:['banana','ma ngo']}) WriteResult({ "nInserted" : 1 })
* db.food.find({fruits:['grapes','mango','apple']}).pretty();

{ "\_id" : 1, "fruits" : [ "grapes", "mango", "apple" ] }

* db.food.find({'fruits.1':'grapes'})
* db.food.find({"fruits":{$size:2}})

{ "\_id" : 3, "fruits" : [ "banana", "mango" ] }

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

{ "\_id" : 1, "fruits" : [ "grapes", "mango" ] }

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

{ "\_id" : 1, "fruits" : [ "grapes", "mango", "apple" ] }

{ "\_id" : 2, "fruits" : [ "grapes", "mango", "cherry" ] }

* db.food.update({\_id:3},{$set:{"fruits.1":"apple"}}) WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
* db.food.update({\_id:2},{$push:{price:{grapes:80,mango:200,cherry:1 00}}}) WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })

>db.Customers.insert({\_custID:1,AcctBal:'100000',AcctType:"saving"});

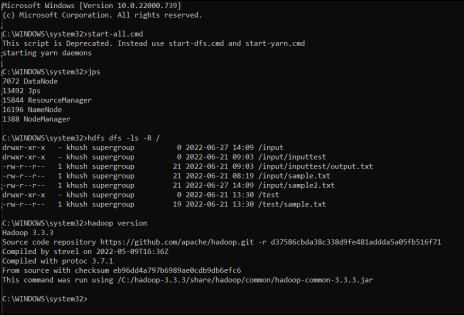
WriteResult({ "nInserted" : 1 })

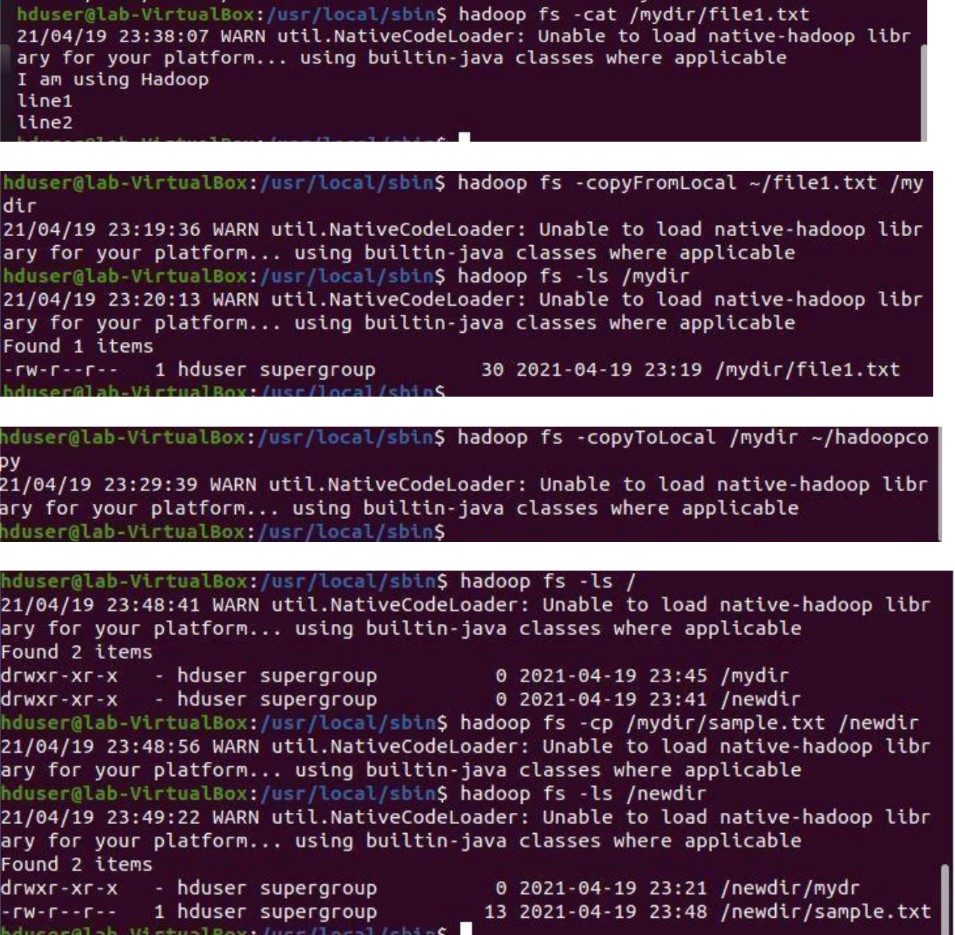
* db.Customers.aggregate({$group:{\_id:"$custID",TotAccBal:{$sum:"$AccBal"}}});

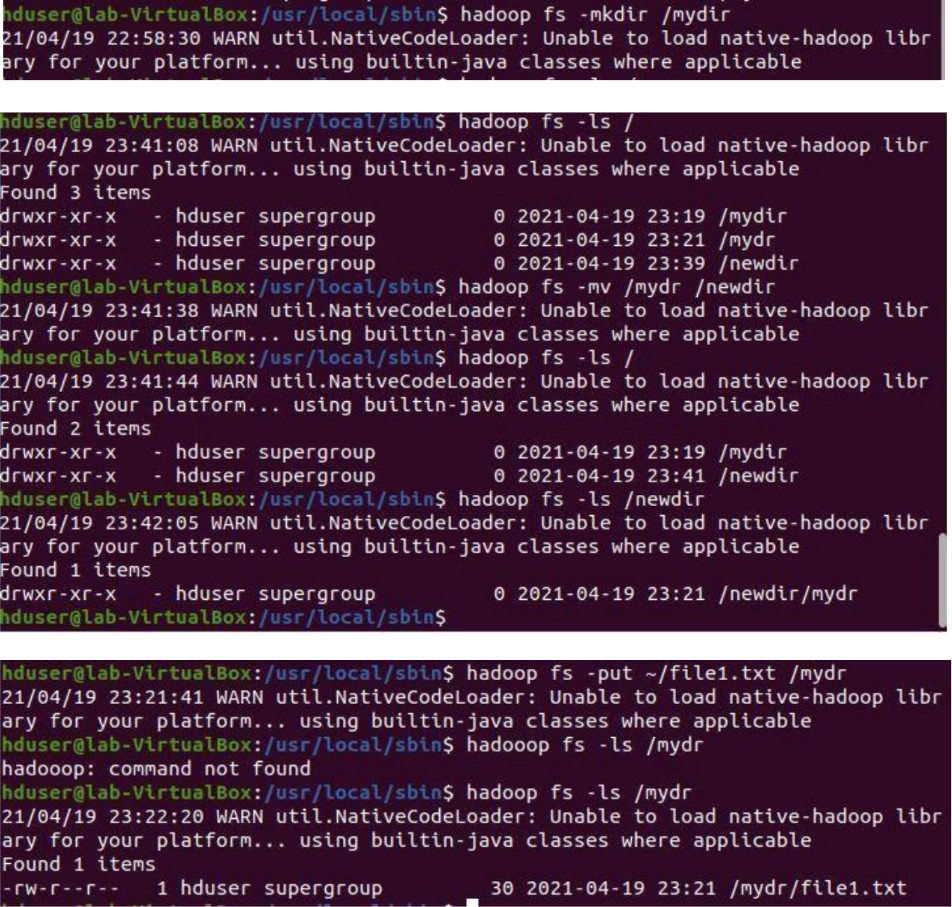
{ "\_id" : null, "TotAccBal" : 0 } db.Customers.aggregate({$match:{AcctType:"saving"}},{$group:{\_id:"$custID",TotAccBal:{$s um:"$AccBal"}}}); { "\_id" : null, "TotAccBal" : 0 }

db.Customers.aggregate({$match:{AcctType:"saving"}},{$group:{\_id:"$custID",TotAccBal:{$su m:"$AccBal"}}},{$ match:{TotAccBal:{$gt:1200}}

# Screenshot of Hadoop installed



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



# Create a Map Reduce program to

1. **find average temperature for each year from the NCDC data set. b) find the mean max temperature for every month**

## 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.FileOutp utFormat; 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));

}

}



## 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.FileInputF ormat; import org.apache.hadoop.mapreduce.lib.output.FileOutp utFormat; 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.c lass);

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;

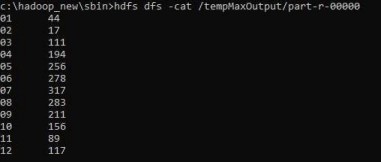
count = 0; days++;

}

}

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

}

}

# 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 Code package wordCount;

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);

}

}

//Mapper Code package wordCount;

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 package wordCount;

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())

{

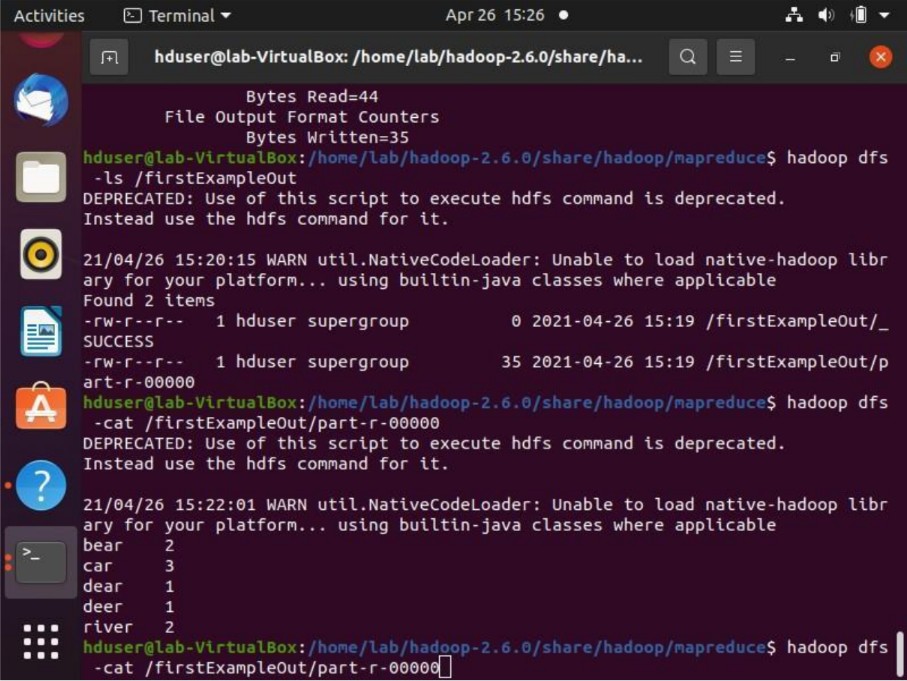
IntWritable i = value.next(); count += i.get();

}

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

}

}



# Create a Map Reduce program to demonstrating join operation

// JoinDriver.java

import org.apache.hadoop.conf.Configured; import org.apache.hadoop.fs.Path;

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

import org.apache.hadoop.mapred.lib.MultipleInputs; import org.apache.hadoop.util.\*;

public class JoinDriver extends Configured implements Tool {

public static class KeyPartitioner implements Partitioner<TextPair, Text> { @Override

public void configure(JobConf job) {} @Override

public int getPartition(TextPair key, Text value, int numPartitions) { return (key.getFirst().hashCode() & Integer.MAX\_VALUE) % numPartitions;

}

}

@Override

public int run(String[] args) throws Exception { if (args.length != 3) {

System.out.println("Usage: <Department Emp Strength input>

<Department Name input>

<output>"); return -1;

}

JobConf conf = new JobConf(getConf(), getClass());

conf.setJobName("Join 'Department Emp Strength input' with 'Department Nameinput'"); 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);

}

} }



# Program to print word count on scala shell and print “Hello world” on scala IDE

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;

# Using RDD and FlaMap count how many times each word appears in a file and write out a list of words whose count is strictly greater than 4 using Spark

val textFile = sc.textFile("/home/bhoom/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()

}

}

