**VISVESVARAYA TECHNOLOGICAL UNIVERSITY “JnanaSangama”, Belgaum -590014, Karnataka.**

****

**LAB REPORT**

**on**

**BIG DATA ANALYTICS**

***Submitted by***

**NEHA CATHRIN A (1BM19CS099)**

***in partial fulfillment for the award of the degree of***

**BACHELOR OF ENGINEERING**

***in***

**COMPUTER SCIENCE AND ENGINEERING**

****

**B.M.S. COLLEGE OF ENGINEERING**

**(Autonomous Institution under VTU)**

**BENGALURU-560019**

**May-2022 to July-2022**

**B. M. S. College of Engineering,**

**Bull Temple Road, Bangalore 560019**

(Affiliated To Visvesvaraya Technological University, Belgaum)

**Department of Computer Science and Engineering**

****

**CERTIFICATE**

This is to certify that the Lab work entitled “**BIG DATA ANALYTICS**” carried out by **NEHA CATHRIN A (1BM19CS099),** who is bonafide student of **B. M. S. College of Engineering.** It is in partial fulfilment 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.

`

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

Assistant Professor Professor and Head

Department of CSE Department of CSE

BMSCE, Bengaluru BMSCE, Bengaluru

INDEX

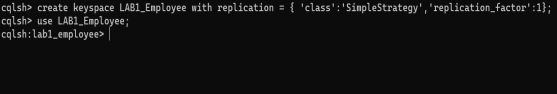
|  |  |
| --- | --- |
| **Sl.**  **No.** | **Experiment Title** |
| **1.** | **Cassandra Lab Program 1: -** Student Database |
| **2.** | **Cassandra Lab Program 2: -** Library Database |
| **3.** | **MongoDB Lab Program 1 (CRUD Demonstration):** |
| **4.** | **HADOOP INSTALLATION** |
| **5.** | **HADOOP SAMPLE** |
| **6.** | **MAPREDUCE TEMPERATURE** |
| **7.** | **MAPREDUCE TOPN** |
| **8.** | **MAPREDUCE JOIN** |
| **9.** | **SCALA INSTALLATION** |
| **10.** | **SCALA WORDCOUNT** |

**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 |

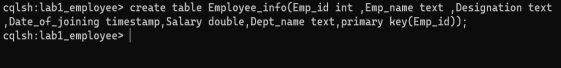
**LAB 1**

1.Create a key space by name Employee

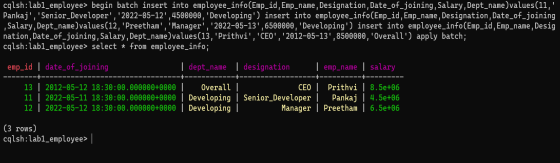


2. Create a column family by name Employee-Info with attributes Emp\_Id Primary Key,

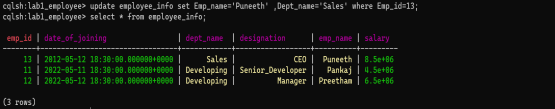
Emp\_Name, Designation, Date\_of\_Joining, Salary, Dept\_Name



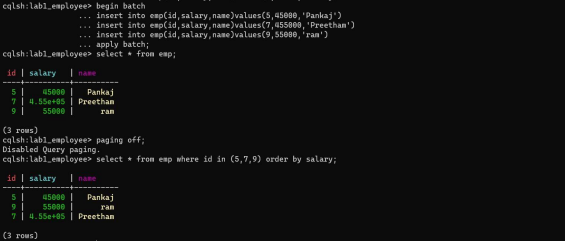
3. Insert the values into the table in batch



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

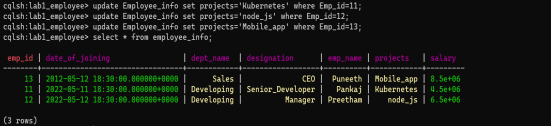


5. Sort the details of Employee records based on salary

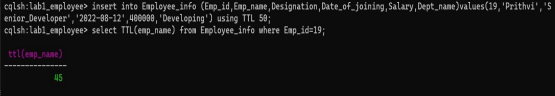


6. Alter the schema of the table Employee\_Info to add a column Projects which stores a set of Projects done by the corresponding Employee.

7. Update the altered table to add project names.

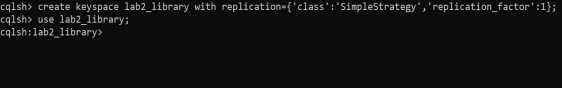


8 Create a TTL of 15 seconds to display the values of Employees.



**LAB 2**

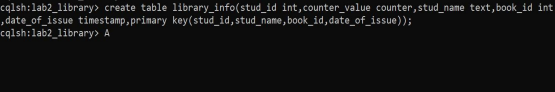
1 Create a key space by name Library

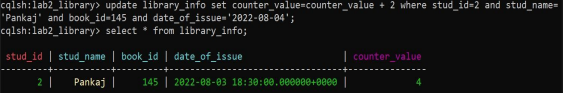


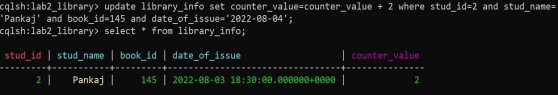
2. Create a column family by name Library-Info with attributes Stud\_Id

Primary Key, Counter\_value of type Counter,

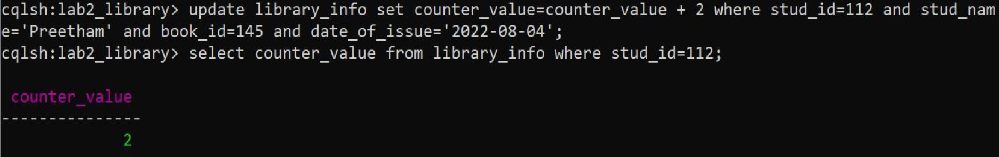
Stud\_Name, Book-Name, Book-Id, Date\_of\_issue

3. Insert the values into the table in batch

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



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



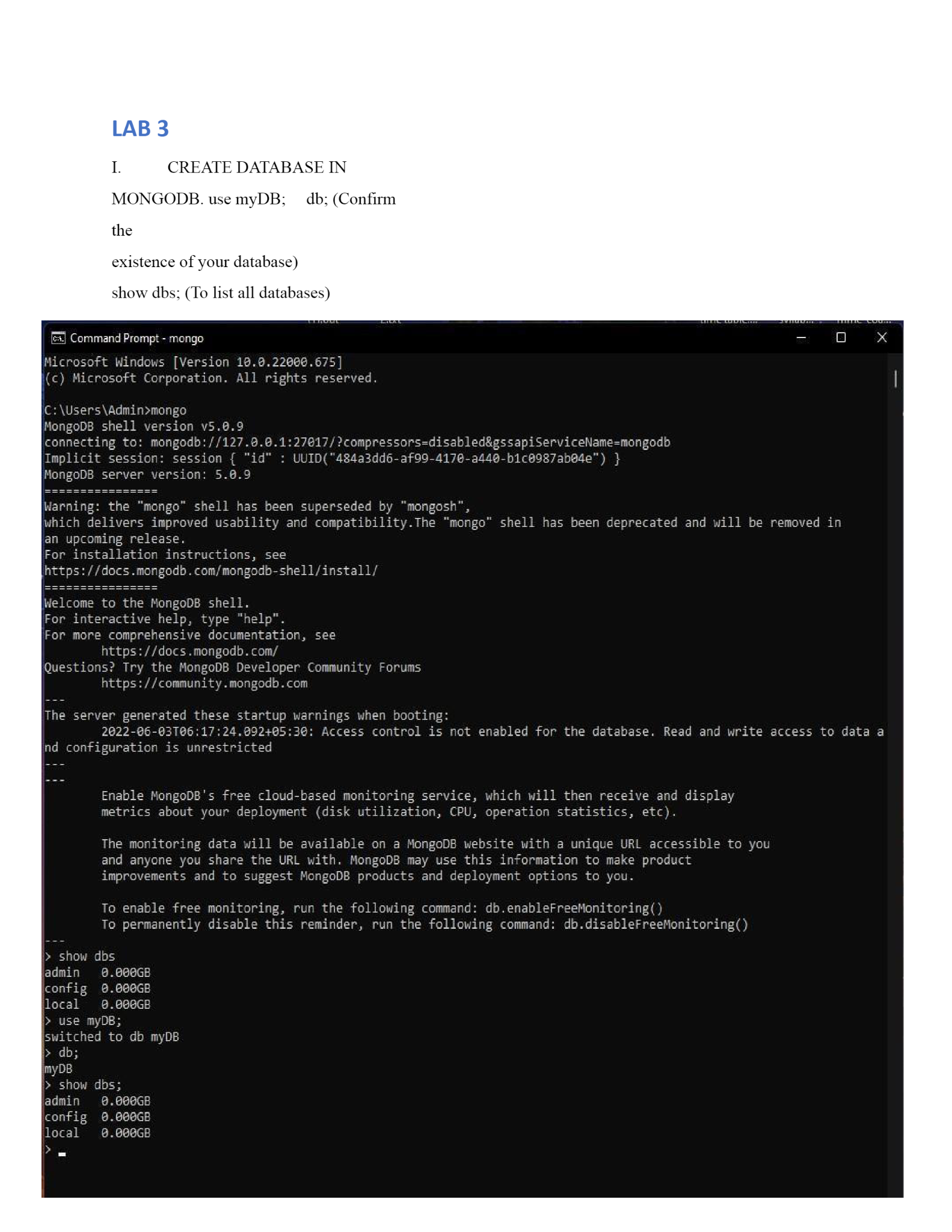
6. Export the created column to a csv file



**LAB 3**

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;Int ernetS 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

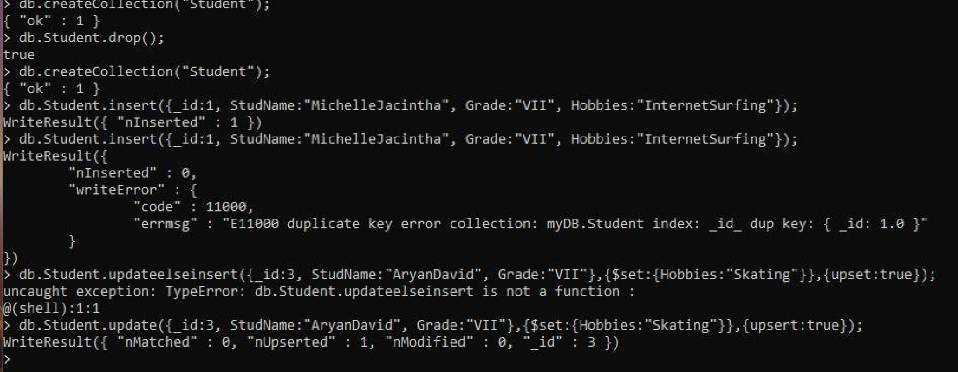
collection, then update the document with new values. (Update his Hobbies

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:{Hobbie s:&quo t;Skatin g&quot;}},{upsert:true});

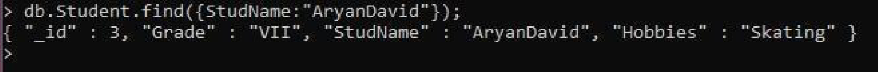


5. FIND METHOD

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

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

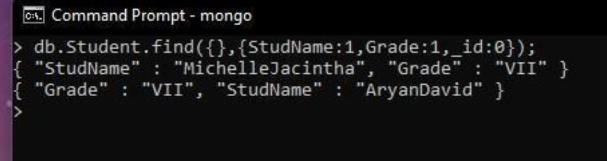
({cond..},{columns.. column:1, columnname:0} )



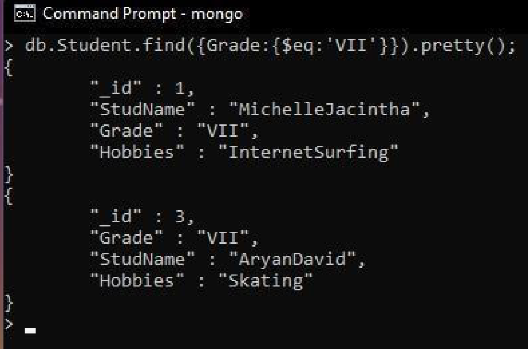
B. To display only the StudName and Grade from all the documents of the

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

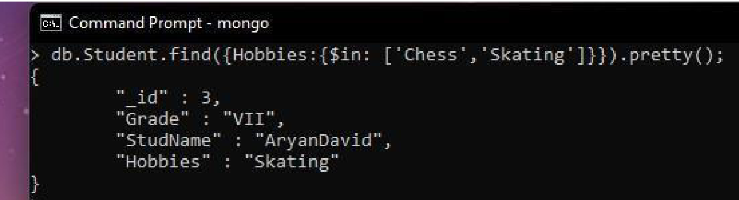
db.Student.find({},{StudName:1,Grade:1,\_id:0});



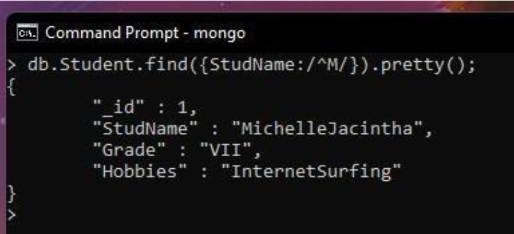
C. To find those documents where the Grade is set to ‘VII’ db.Student.find({Grade:{$eq:&#39;VII&#39;}}).pretty();



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 ();

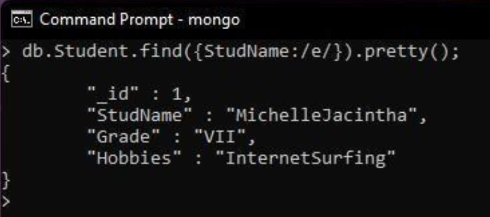


E. To find documents from the Students collection where the StudName begins with “M”. db.Student.find({StudName:/^M/}).pretty();

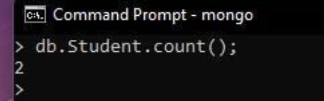


F. To find documents from the

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();



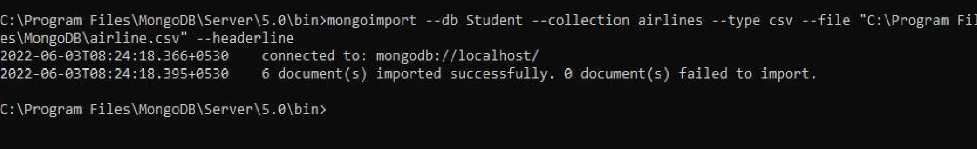
III. Import data from a CSV file

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

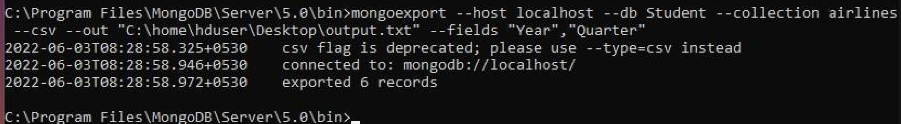
from

“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”

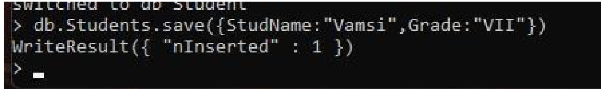


V. Save Method :

Save() method will insert a new document, if the document with the \_id does

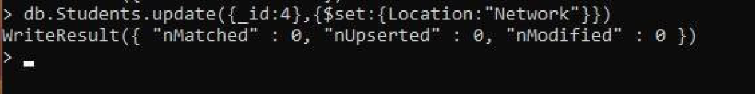
not exist. If it exists it will replace the exisiting document.

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



VI. Add a new field to existing Document:

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



VII.. 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

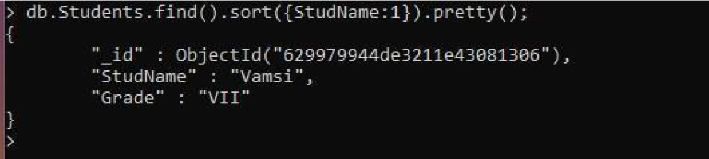
document in Ascending order

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

desending order : db.Students.find().sort({StudName:-

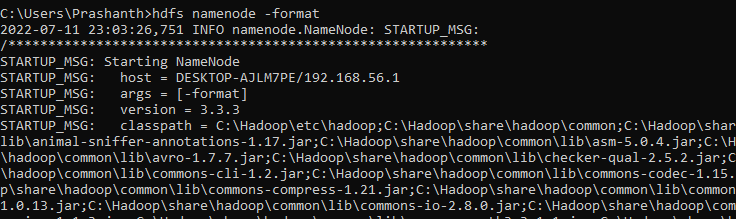
1}).pretty(); to Skip the 1 st two documents from the

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



**LAB 4**

SCREENSHOT OF HADOOP INSTALLATION



**LAB 5**

Execution of HDFS Commands for interaction with Hadoop Environment. (Minimum 10 commands to be executed)

c:\hadoop\_new\sbin>hdfs dfs -mkdir /temp

c:\hadoop\_new\sbin>hdfs dfs -copyFromLocal E:\Desktop\sample.txt \temp c:\hadoop\_new\sbin>hdfs dfs -ls \temp

Found 1 items

-rw-r--r-- 1 Admin supergroup 11 2021-06-11 21:12 /temp/sample.txt

c:\hadoop\_new\sbin>hdfs dfs -cat \temp\sample.txt hello world

c:\hadoop\_new\sbin>hdfs dfs -get \temp\sample.txt E:\Desktop\temp

c:\hadoop\_new\sbin>hdfs dfs -put E:\Desktop\temp \temp

c:\hadoop\_new\sbin>hdfs dfs -ls \temp Found 2 items

-rw-r--r-- 1 Admin supergroup 11 2021-06-11 21:12 /temp/sample.txt

drwxr-xr-x - Admin supergroup 0 2021-06-11 21:15 /temp/temp c:\hadoop\_new\sbin>hdfs dfs -mv \lab1 \temp

c:\hadoop\_new\sbin>hdfs dfs -ls \temp Found 3 items

drwxr-xr-x - Admin supergroup 0 2021-04-19 15:07 /temp/lab1

-rw-r--r-- 1 Admin supergroup 11 2021-06-11 21:12 /temp/sample.txt

drwxr-xr-x - Admin supergroup 0 2021-06-11 21:15 /temp/temp

c:\hadoop\_new\sbin>hdfs dfs -rm /temp/sample.txt Deleted /temp/sample.txt

c:\hadoop\_new\sbin>hdfs dfs -ls \temp Found 2 items

drwxr-xr-x - Admin supergroup 0 2021-04-19 15:07 /temp/lab1

drwxr-xr-x - Admin supergroup 0 2021-06-11 21:15 /temp/temp c:\hadoop\_new\sbin>hdfs dfs -copyFromLocal E:\Desktop\sample.txt \temp

c:\hadoop\_new\sbin>hdfs dfs -ls \temp Found 3 items

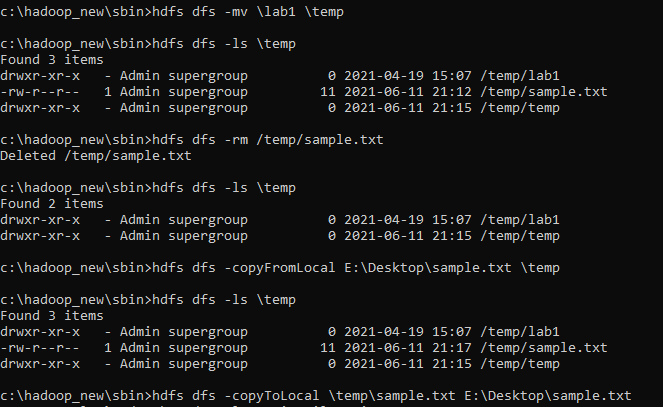
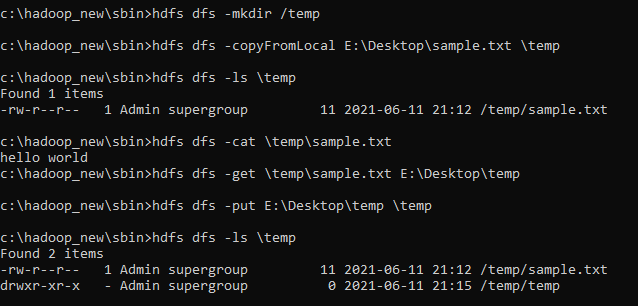
drwxr-xr-x - Admin supergroup 0 2021-04-19 15:07 /temp/lab1

-rw-r--r-- 1 Admin supergroup 11 2021-06-11 21:17 /temp/sample.txt

drwxr-xr-x - Admin supergroup 0 2021-06-11 21:15 /temp/temp

c:\hadoop\_new\sbin>hdfs dfs -copyToLocal \temp\sample.txt E:\Desktop\sample.txt

SCREENSHOTS –



**LAB 6**

Map reduce Temperature

For the given file, Create a Map Reduce program to

1. Find the average temperature for each year from the NCDC data set.

// AverageDriver.java package temperature;

import org.apache.hadoop.io.\*; import org.apache.hadoop.fs.\*;

import org.apache.hadoop.mapreduce.\*;

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.java package temperature;

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

import org.apache.hadoop.mapreduce.\*; import java.io.IOException;

public class AverageMapper extends Mapper <LongWritable, Text, Text, IntWritable>

{

public static final int MISSING = 9999;

public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException

{

String line = value.toString(); String year = line.substring(15,19); int temperature;

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 != MISSING && quality.matches("[01459]")) context.write(new Text(year),new IntWritable(temperature));

}

}

//AverageReducer.java package temperature;

import org.apache.hadoop.io.IntWritable; import org.apache.hadoop.io.Text; import org.apache.hadoop.mapreduce.\*; import java.io.IOException;

public class AverageReducer extends Reducer <Text, IntWritable,Text, IntWritable>

{

public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException,InterruptedException

{

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

for (IntWritable value : values)

{

max\_temp += value.get(); count+=1;

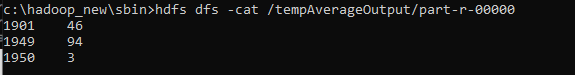
}

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

}

}

SCREENSHOT -



1. Find the mean max temperature for every month.

//TempDriver.java package temperatureMax;

import org.apache.hadoop.io.\*; import org.apache.hadoop.fs.\*;

import org.apache.hadoop.mapreduce.\*;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat; import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;

public class TempDriver

{

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(TempDriver.class); job.setJobName("Max temperature"); FileInputFormat.addInputPath(job,new Path(args[0])); FileOutputFormat.setOutputPath(job,new Path (args[1]));

job.setMapperClass(TempMapper.class);

job.setReducerClass(TempReducer.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(IntWritable.class);

System.exit(job.waitForCompletion(true)?0:1);

}

}

//TempMapper.java package temperatureMax;

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

import org.apache.hadoop.mapreduce.\*; import java.io.IOException;

public class TempMapper extends Mapper <LongWritable, Text, Text, IntWritable>

{

public static final int MISSING = 9999;

public void map(LongWritable key, Text value, Context context) throws IOException, InterruptedException

{

String line = value.toString();

String month = line.substring(19,21); int temperature;

if (line.charAt(87)=='+')

temperature = Integer.parseInt(line.substring(88, 92));

else

temperature = Integer.parseInt(line.substring(87, 92));

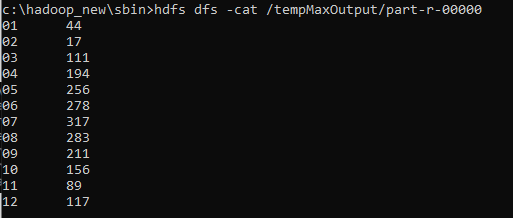
String quality = line.substring(92, 93);

if(temperature != MISSING && quality.matches("[01459]")) context.write(new Text(month),new IntWritable(temperature));

}

}

SCREENSHOT -



**LAB 7**

Map reduce TOPN

For a given Text file, create a Map Reduce program to sort the content in an alphabetic order listing only top ‘n’ maximum occurrence of words.

// TopN.java package sortWords;

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.Reducer;

import org.apache.hadoop.mapreduce.lib.input.FileInputFormat; import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat; import org.apache.hadoop.util.GenericOptionsParser;

import utils.MiscUtils;

import java.io.IOException; import java.util.\*;

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.setCombinerClass(TopNReducer.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);

}

/\*\*

* The mapper reads one line at the time, splits it into an array of single words and emits every
* word to the reducers with the value of 1.

\*/

public static class TopNMapper extends Mapper<Object, Text, Text, IntWritable> {

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

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

@Override

public void map(Object key, Text value, Context context) throws IOException, InterruptedException {

String cleanLine = value.toString().toLowerCase().replaceAll(tokens, " "); StringTokenizer itr = new StringTokenizer(cleanLine);

while (itr.hasMoreTokens()) {

word.set(itr.nextToken().trim()); context.write(word, one);

}

}

}

/\*\*

* The reducer retrieves every word and puts it into a Map: if the word already exists in the
* map, increments its value, otherwise sets it to 1.

\*/

public static class TopNReducer extends Reducer<Text, IntWritable, Text, IntWritable> { private Map<Text, IntWritable> countMap = new HashMap<>();

@Override

public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {

// computes the number of occurrences of a single word int sum = 0;

for (IntWritable val : values) { sum += val.get();

}

// puts the number of occurrences of this word into the map.

// We need to create another Text object because the Text instance

// we receive is the same for all the words countMap.put(new Text(key), new IntWritable(sum));

}

@Override

protected void cleanup(Context context) throws IOException, InterruptedException { Map<Text, IntWritable> sortedMap = MiscUtils.sortByValues(countMap);

int counter = 0;

for (Text key : sortedMap.keySet()) { if (counter++ == 3) {

break;

}

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

}

}

}

/\*\*

* The combiner retrieves every word and puts it into a Map: if the word already exists in the
* map, increments its value, otherwise sets it to 1.

\*/

public static class TopNCombiner extends Reducer<Text, IntWritable, Text, IntWritable> {

@Override

public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException, InterruptedException {

// computes the number of occurrences of a single word int sum = 0;

for (IntWritable val : values) { sum += val.get();

}

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

}

}

}

// MiscUtils.java package utils;

import java.util.\*; public class MiscUtils {

/\*\*

* sorts the map by values. Taken from:
* <http://javarevisited.blogspot.it/2012/12/how-to-sort-hashmap-java-by-key-and-value.html>

\*/

public static <K extends Comparable, V extends Comparable> Map<K, V> sortByValues(Map<K, V> map) {

List<Map.Entry<K, V>> entries = new LinkedList<Map.Entry<K, V>>(map.entrySet()); Collections.sort(entries, new Comparator<Map.Entry<K, V>>() {

@Override

public int compare(Map.Entry<K, V> o1, Map.Entry<K, V> o2) { return o2.getValue().compareTo(o1.getValue());

}

});

//LinkedHashMap will keep the keys in the order they are inserted

//which is currently sorted on natural ordering

Map<K, V> sortedMap = new LinkedHashMap<K, V>(); for (Map.Entry<K, V> entry : entries) {

sortedMap.put(entry.getKey(), entry.getValue());

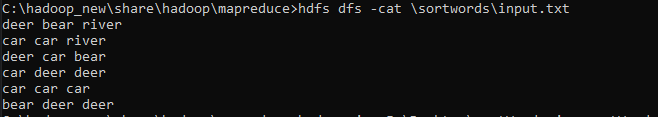
}

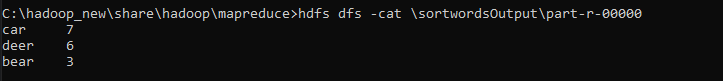
return sortedMap;

}

}

SCREENSHOTS –





**LAB 8**

Create a Hadoop Map Reduce program to combine information from the users file along with Information from the posts file by using the concept of join and display user\_id, Reputation and Score.

// 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) {}

numPartitions;

@Override

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

}

}

@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;

}

input'");

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

conf.setJobName("Join 'Department Emp Strength input' with 'Department Name

Path AInputPath = new Path(args[0]); Path BInputPath = new Path(args[1]); Path outputPath = new Path(args[2]);

Posts.class); User.class);

MultipleInputs.addInputPath(conf, AInputPath, TextInputFormat.class, MultipleInputs.addInputPath(conf, BInputPath, TextInputFormat.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);

}

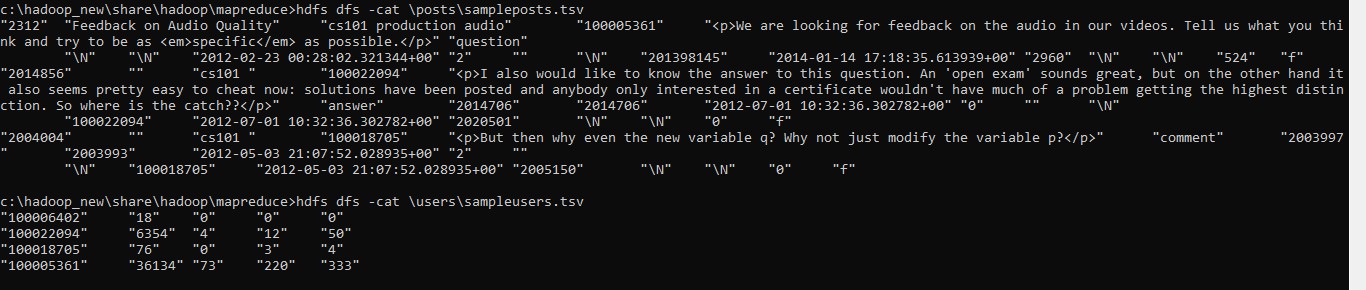
return super.compare(a, b);

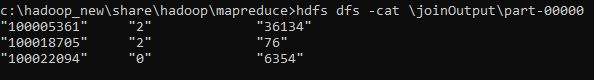
}

}

}

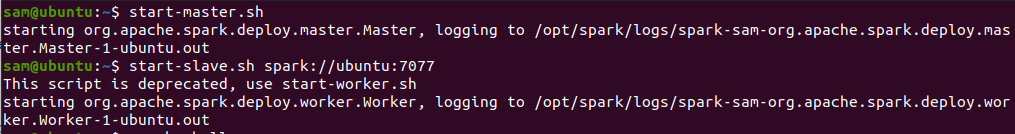
SCREENSHOTS –

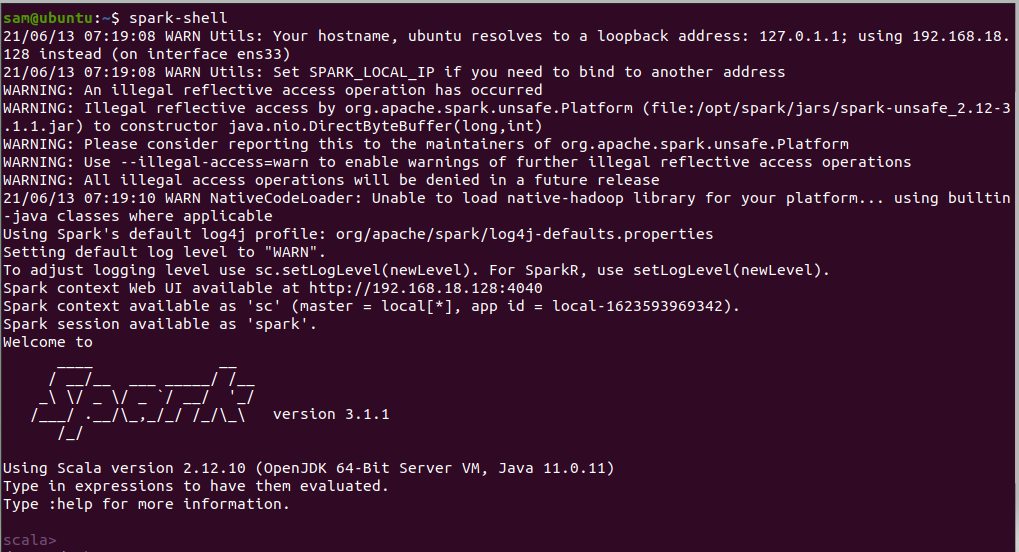




**LAB 9:**

Scala installation screenshot





**LAB 10**

wordcount

// scala shell

scala> val textfile = sc.textFile("/home/sam/Desktop/abc.txt")

textfile: org.apache.spark.rdd.RDD[String] = /home/sam/Desktop/abc.txt MapPartitionsRDD[1] at textFile at <console>:24

scala> val counts = textfile.flatMap(line => line.split(" ")).map(word => (word,1)).reduceByKey(+)

counts: org.apache.spark.rdd.RDD[(String, Int)] = ShuffledRDD[4] at reduceByKey at

<console>:25

scala> import scala.collection.immutable.ListMap import scala.collection.immutable.ListMap

scala> val sorted = ListMap(counts.collect.sortWith(.\_2>.2):\*)

scala> println(sorted)

ListMap(car -> 7, deer -> 5, bear -> 3, river -> 3, -> 1)

scala> for((k,v)<-sorted)

| {

| if(v>4)

| {

| println(k+"-"+v)

| }

| } car-7 deer-5



