# 

# **INDEX**

| **Sr. No** | **Assignment Title** | **Date** | **Page No** |
| --- | --- | --- | --- |
| 1 | Write a program to find occurrence of any specific word | 08-09-2022 | 3-7 |
| 2 | Write a program to find average word length. | 10-08-2022 | 8-11 |
| 3 | Encrypt the email ids | 23-08-2022 | 12-14 |
| 4 | Multiple file input demo assignment | 29-08-2022 | 15-19 |
| 5 | Web Log Analysis - partitioner demo assignment  I. Odd roll no  II. Even roll No | 30-08-2022 | 20-34 |
| 6 | Partitioner Demo - Cust dataset | 02-09-2022 | 35-41 |
| 7 | Government Crime Assignment - Pig | 08-09-2022 | 42-46 |

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# **Assignment : 01**

# 

# **Write a program to find occurrence of any specific word**

**Code:**

import java.io.IOException;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

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

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

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

public class Count {

public static class WordMapper extends Mapper<LongWritable,Text,Text,IntWritable>{

@Override

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

throws IOException, InterruptedException{

String line = value.toString();

String find = "big";

for (String word : line.split("\\W+")){

if(find.equals(word.toLowerCase())){

context.write(new Text(find), new IntWritable(1));

}

}

}

}

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

@Override

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

int wordCount = 0;

for (IntWritable value: values){

wordCount += value.get();

}

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

}

}

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

if(args.length != 2){

System.out.printf("Usage: WordCount <input dir> <output dir>\n");

System.exit(-1);

}

Job job = new Job();

job.setJarByClass(Count.class);g

job.setJobName("WordCount");

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

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

job.setMapperClass(WordMapper.class);

job.setReducerClass(SumReducer.class);

job.setOutputKeyClass(Text.class);

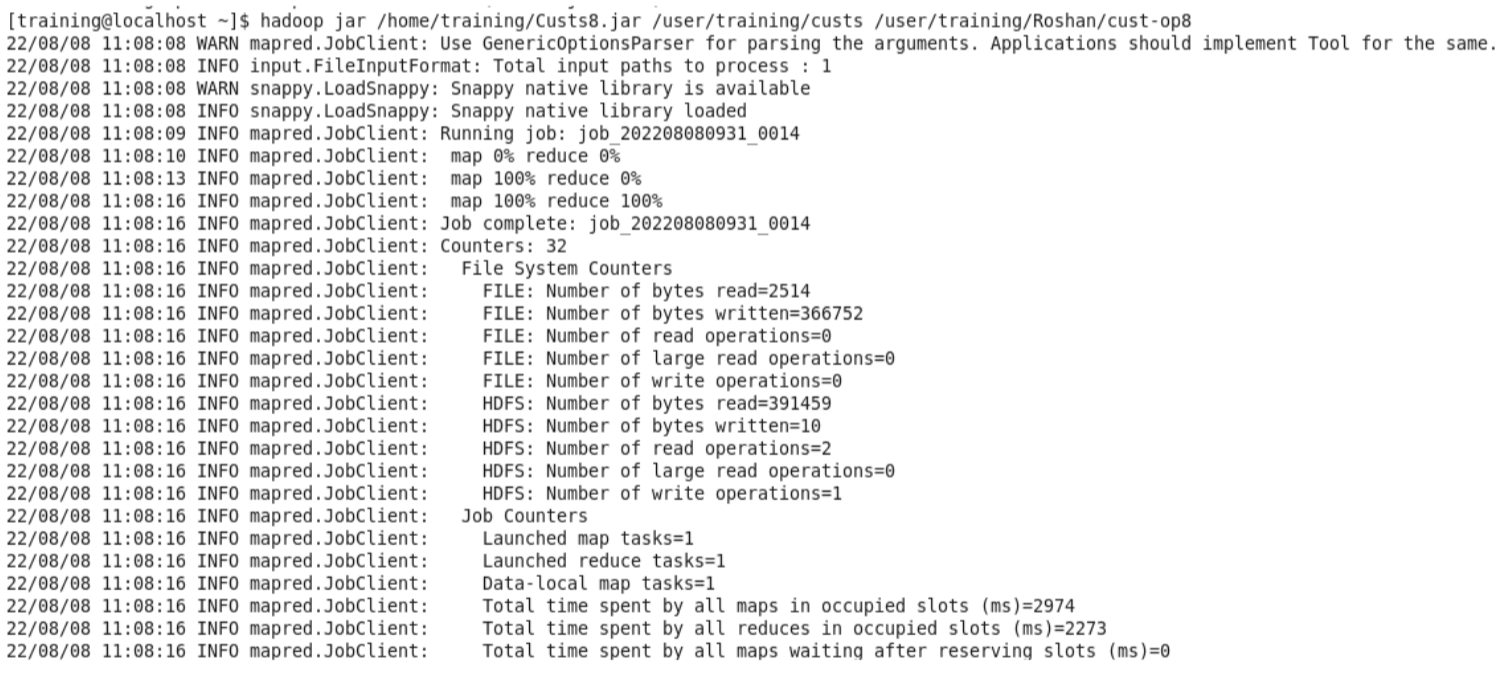
job.setOutputValueClass(IntWritable.class);

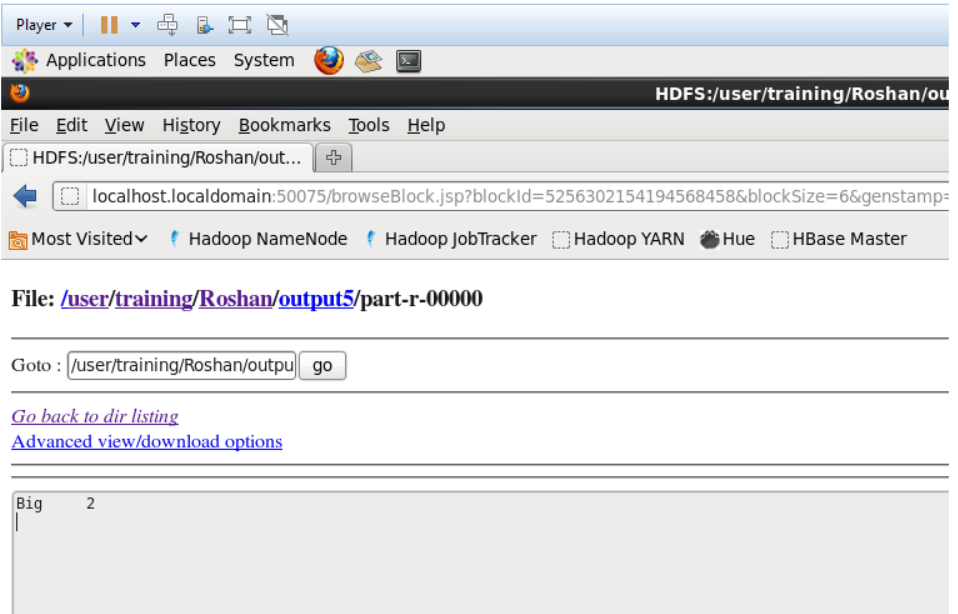
boolean success = job.waitForCompletion(true);

System.exit(success ? 0 : 1);

}

}





**Count of specific profession**

**Code:**

import java.io.IOException;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.Text;

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

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

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

public class Cust {

public static class WordMapper extends Mapper<LongWritable,Text,Text,IntWritable>{

@Override

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

throws IOException, InterruptedException{

String line = value.toString();

String find = "Pilot";

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

if(find.equals(word)){

context.write(new Text(find), new IntWritable(1));

}

}

}

}

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

@Override

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

int wordCount = 0;

for (IntWritable value: values){

wordCount += value.get();

}

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

}

}

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

if(args.length != 2){

System.out.printf("Usage: WordCount <input dir> <output dir>\n");

System.exit(-1);

}

Job job = new Job();

job.setJarByClass(Cust.class);

job.setJobName("Custs");

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

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

job.setMapperClass(WordMapper.class);

job.setReducerClass(SumReducer.class);

job.setOutputKeyClass(Text.class);

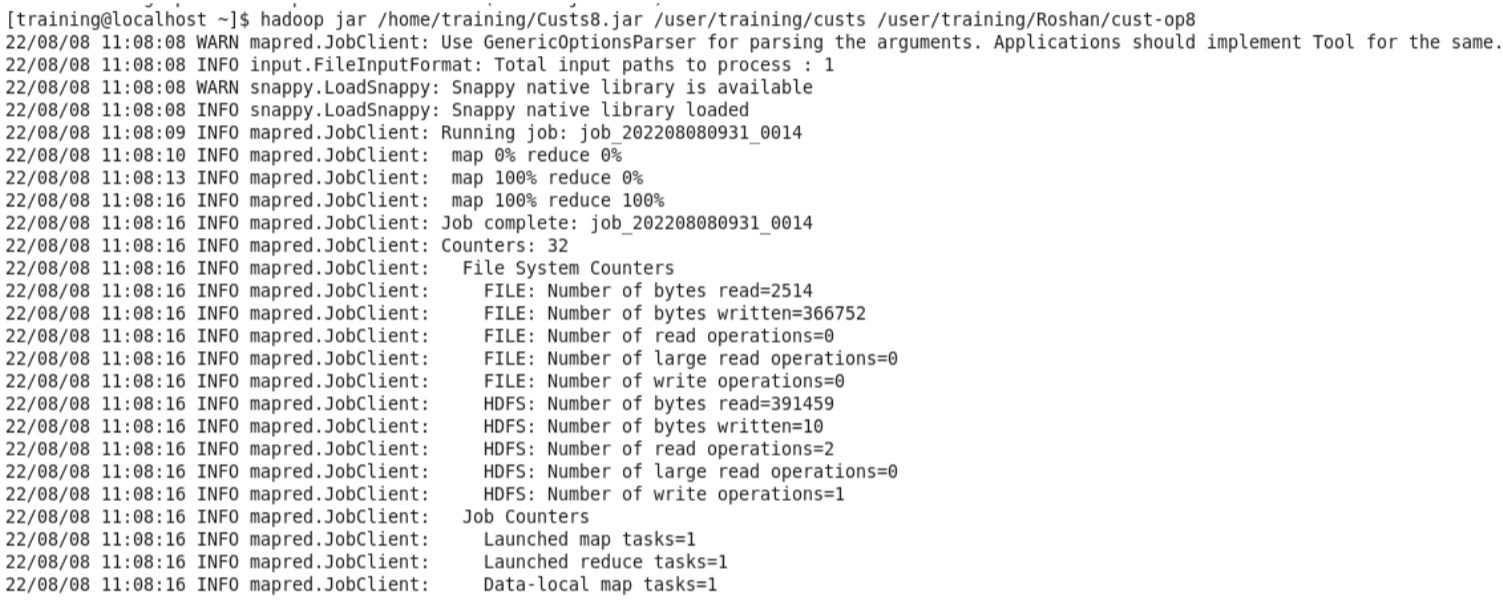
job.setOutputValueClass(IntWritable.class);

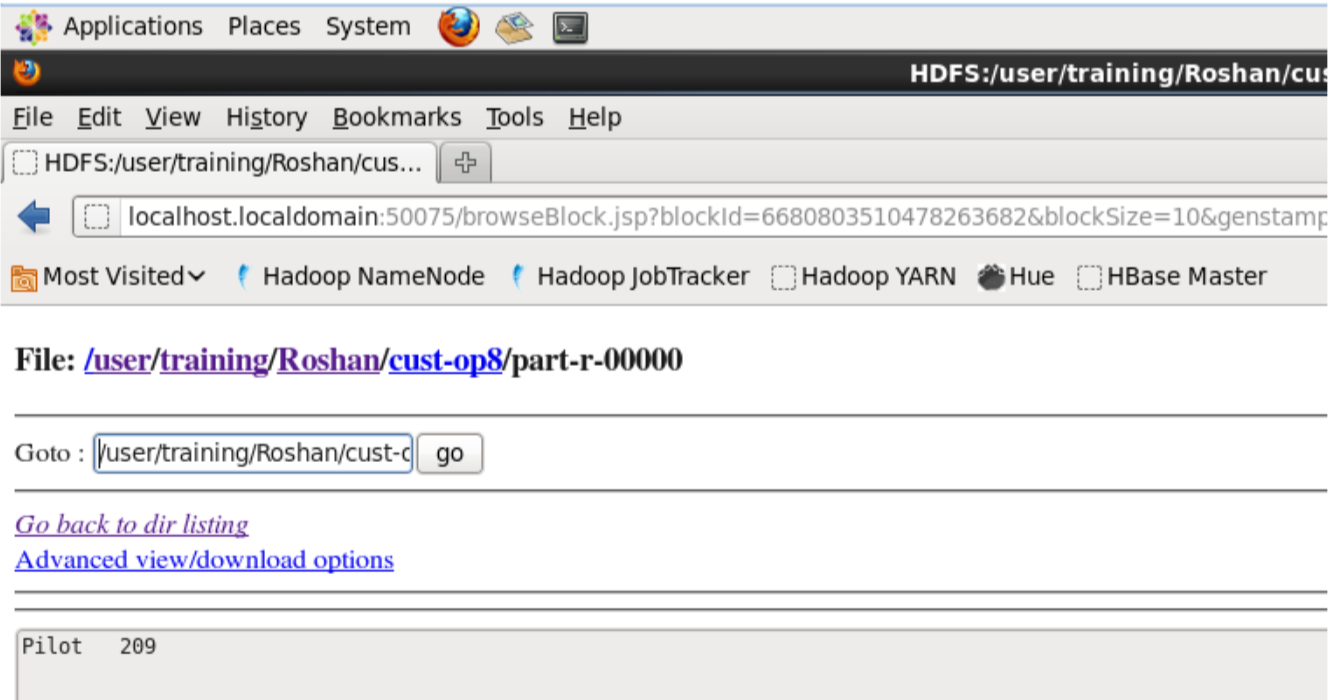
boolean success = job.waitForCompletion(true);

System.exit(success ? 0 : 1);

}

}





# **Assignment : 02**

# 

# **Write a program to find average word length.**

**Code:**

import java.io.IOException;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.FloatWritable;

import org.apache.hadoop.io.Text;

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

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

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.io.LongWritable;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.mapreduce.Reducer;

public class Average {

public static class WordMapper extends Mapper <LongWritable, Text, Text, FloatWritable>

{

@Override

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

throws IOException, InterruptedException{

Text firstLetter = new Text();

FloatWritable wordLength = new FloatWritable();

String line = value.toString().toLowerCase();

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

if (word.length() > 0) {

firstLetter.set(String.valueOf(word.charAt(0)));

wordLength.set(word.length());

context.write(firstLetter, wordLength);

}

}

}

}

public static class SumReducer extends Reducer<Text,FloatWritable,Text,FloatWritable>{

public void reduce(Text key,Iterable<FloatWritable>values,Context context)

throws IOException,InterruptedException{

int sum =0;

int count =0;

int Average =0;

for(FloatWritable val:values) {

sum += val.get();

count = count+1;

}

Average = sum/count;

context.write(key,new FloatWritable(Average));

}

}

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

if(args.length != 2){

System.out.printf(

"Usage: WordCount <input dir> <output dir> \n");

System.exit(-1);

}

Job job = new Job();

job.setJarByClass(Average.class);

job.setJobName("Word Count");

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

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

job.setMapperClass(WordMapper.class);

job.setReducerClass(SumReducer.class);

job.setOutputKeyClass(Text.class);

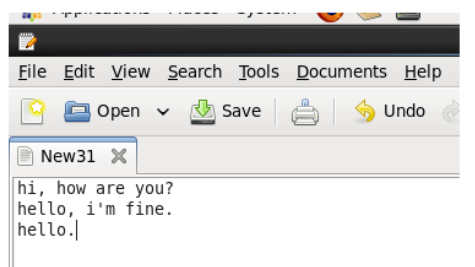
job.setOutputValueClass(FloatWritable.class);

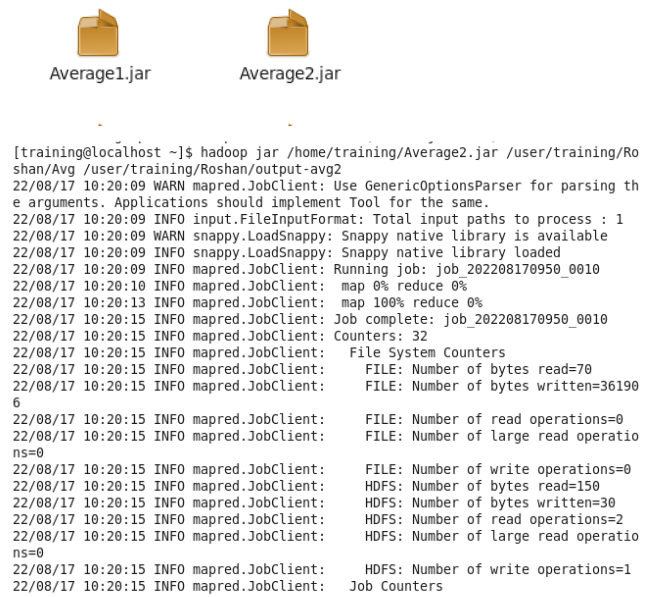
boolean success = job.waitForCompletion(true);

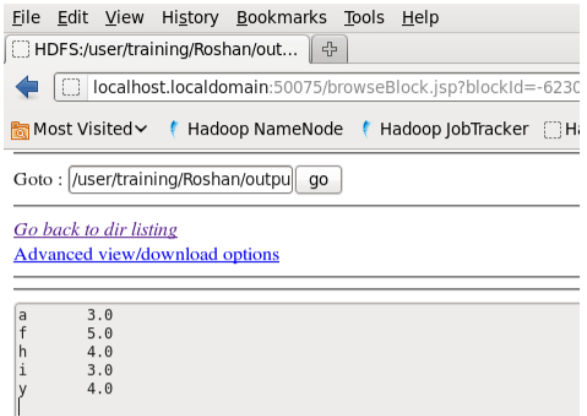
System.exit(success ? 0 :1);

}

}







# **Assignment : 03**

# 

# **Encrypt the email ids**

**use "Only mapper" design pattern**

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

import org.apache.hadoop.io.LongWritable;

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

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

import org.apache.hadoop.util.GenericOptionsParser;

public class Encry {

public static class MapForEncryption extends Mapper<LongWritable, Text, Text, Text>

{

public void map(LongWritable key, Text Value, Context con)throws IOException, InterruptedException

{

String line=Value.toString();

int encry; //created a object which will store the encrypted character

String mask = ""; //mask object initialized with null value which will store the actual output text

for(int i=0;i<line.length();i++) //for loop which will iterate through line

{

char c = line.charAt(i); //store each character at index i

encry = 4 + (int)c; //increment the alphabet by 4

c = (char)encry; //type cast encry into char

mask += c; //store the final text in mask

}

//String line1 = line.replace("||",",");

Text outputKey = new Text(mask);

Text outputValue = new Text("");

con.write(outputKey,outputValue);

}

}

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

Configuration c = new Configuration();

String[] files = new GenericOptionsParser(c,args).getRemainingArgs();

Path input = new Path(files[0]);

Path output = new Path(files[1]);

Job j = new Job(c,"FormatEmpFile");

j.setJarByClass(Encry.class);

j.setJobName("");

j.setMapperClass(MapForEncryption.class);

j.setNumReduceTasks(0);

j.setMapOutputKeyClass(Text.class);

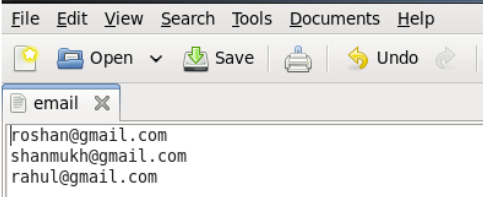
j.setMapOutputValueClass(Text.class);

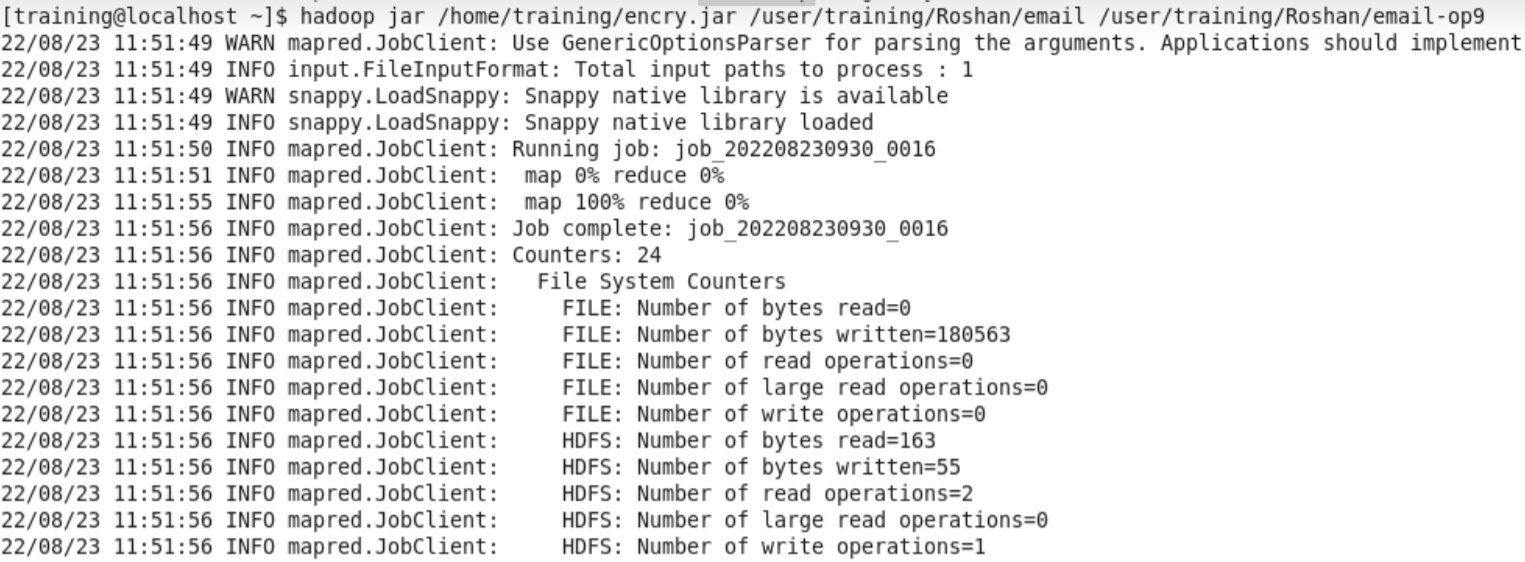
FileInputFormat.addInputPath(j,input);

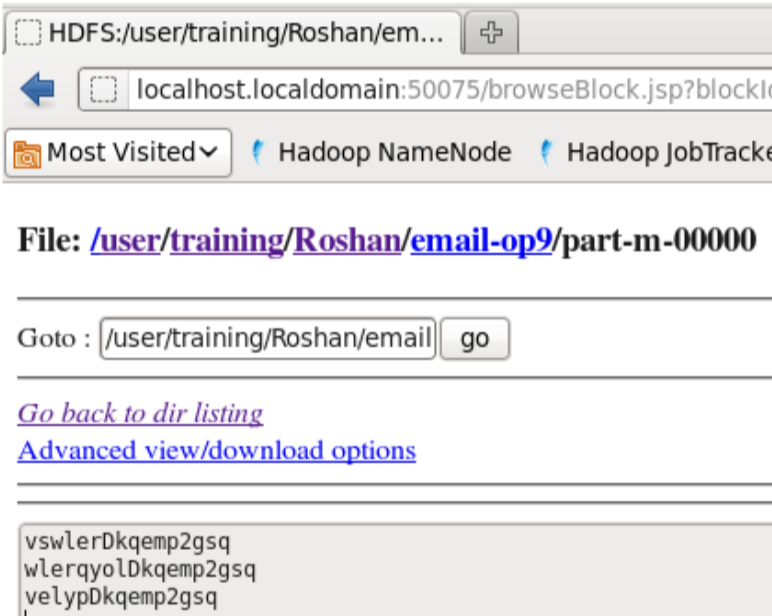
FileOutputFormat.setOutputPath(j,output);

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

}}







# **Assignment : 04**

# Multiple file input demo assignment

**Create 2 or 3 input files on your own , in which the data is present in different formats. Write a program to process these files using different map classes and perform any one aggregate function like sum, max, min etc. on it.**

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.IntWritable;

import org.apache.hadoop.io.LongWritable;

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.MultipleInputs;

import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;

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

import org.apache.hadoop.util.GenericOptionsParser;

public class MultipleInputFiles {

public static class map1 extends Mapper<LongWritable,Text,Text,IntWritable>

{

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

{

String line = value.toString();

String[] line1 = line.split(",");

String name = line1[0];

Text outputKey = new Text(name);

int cost = Integer.parseInt(line1[2]);

IntWritable outputValue = new IntWritable(cost);

con.write(outputKey, outputValue);

}

}

public static class Map2 extends Mapper<LongWritable, Text, Text, IntWritable>

{

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

{

String line = value.toString();

String[] line1 = line.split(",");

String name = line1[0];

Text outputKey = new Text(name);

int cost = Integer.parseInt(line1[2]);

IntWritable outputValue = new IntWritable(cost);

con.write(outputKey, outputValue);

}

}

public static class Map3 extends Mapper<LongWritable, Text, Text, IntWritable>

{

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

{

String line = value.toString();

String[] line1 = line.split(",");

String name = line1[1];

Text outputKey = new Text(name);

int cost = Integer.parseInt(line1[3]);

IntWritable outputValue = new IntWritable(cost);

con.write(outputKey, outputValue);

}

}

public static class Red extends Reducer<Text, IntWritable, Text,IntWritable>

{

public void reduce(Text name, Iterable<IntWritable>total\_cost, Context con) throws IOException, InterruptedException

{

//to get the sum of cost

// int sum = 0;

// for(IntWritable value : total\_cost){

// sum += value.get();

//}

//to get the max cost

int max = 0;

for (IntWritable value : total\_cost) {

if (value.get() > max) {

max = value.get();

}

}

con.write(name,new IntWritable(max));

}

}

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

Configuration c = new Configuration();

GenericOptionsParser parser = new GenericOptionsParser(c,args);

String[] files = parser.getRemainingArgs();

Path p1 = new Path(files[0]);

Path p2 = new Path(files[1]);

Path p3 = new Path(files[2]);

Path p4 = new Path(files[3]);

Job j = new Job(c,"multiple");

j.setJarByClass(MultipleInputFiles.class);

j.setMapperClass(map1.class);

j.setMapperClass(Map2.class);

j.setReducerClass(Red.class);

j.setOutputKeyClass(Text.class);

j.setOutputValueClass(IntWritable.class);

MultipleInputs.addInputPath(j,p1,TextInputFormat.class,map1.class);

MultipleInputs.addInputPath(j, p2, TextInputFormat.class,Map2.class);

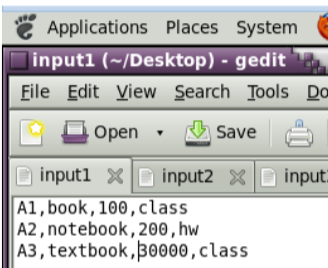
MultipleInputs.addInputPath(j, p3, TextInputFormat.class,Map3.class);

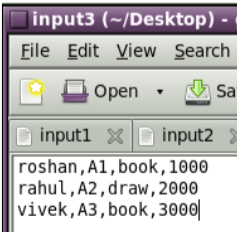
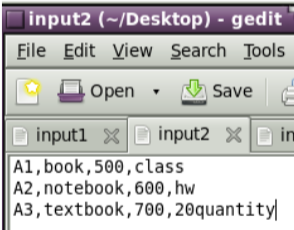
FileOutputFormat.setOutputPath(j,p4);

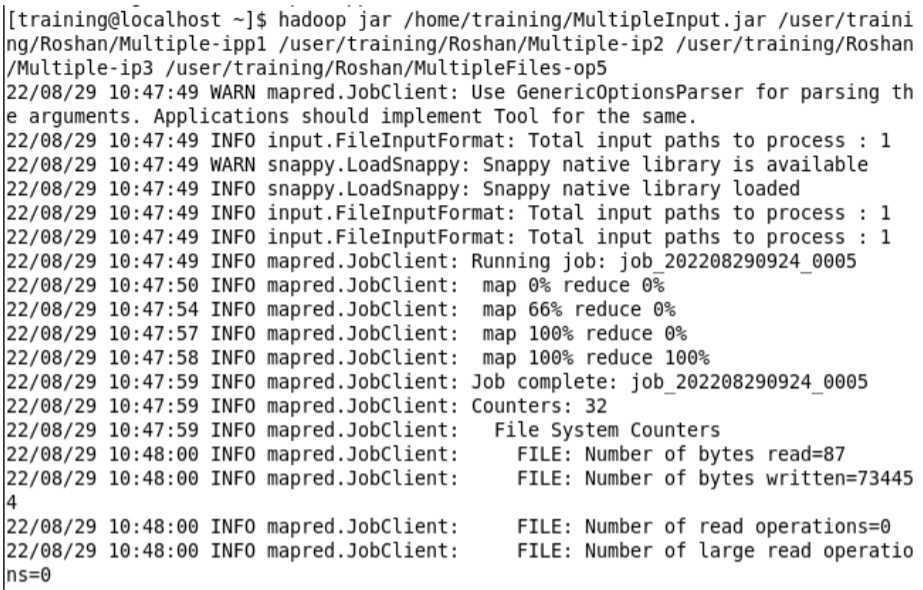
System.exit(j.waitForCompletion(true)?0:1);

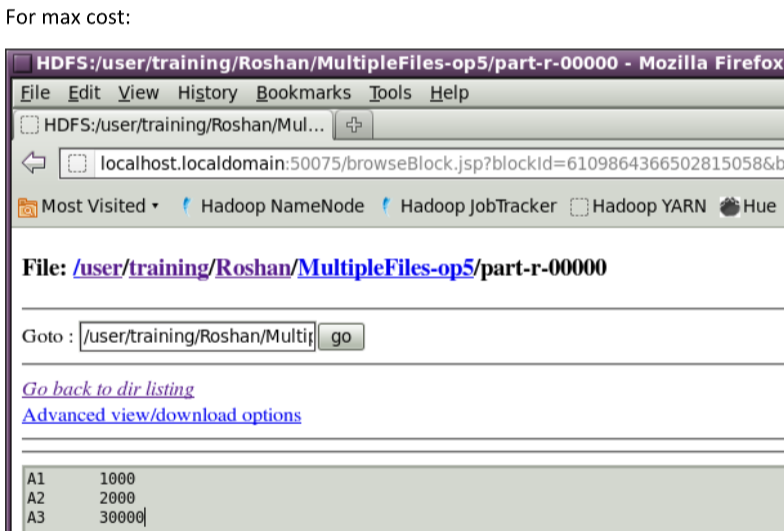
}

}









# **Assignment : 05**

# 

# **Web Log Analysis - partitioner demo assignment**

Odd

Write a MapReduce program to find profession-wise (choose any 5 professions) of customer count from the "cust" dataset.

Code:

**import** java.io.\*;

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

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

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

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

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

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

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

**public** **class** part **extends** Configured **implements** Tool{

//map class

**public** **static** **class** Map **extends** Mapper<LongWritable, Text, Text, IntWritable> {

@Override **public** **void** map (LongWritable key, Text value, Context context)

**throws** IOException, InterruptedException {

String line = value.toString();

String[] date =line.split("/");

String month=date[1];

context.write(**new** Text (month), **new** IntWritable (1));

}

}

//Reducer class

**public** **static** **class** SumReducer **extends** Reducer<Text,IntWritable,Text,IntWritable> {

@Override **public** **void** reduce(Text name, Iterable<IntWritable>values,Context context)

**throws** IOException, InterruptedException {

**int** Count = 0;

**for** (IntWritable value : values) {

Count += value.get();

}

context.write(name, **new** IntWritable (Count));

}

}

//Partitioner class

**public** **static** **class** CaderPartitioner **extends** Partitioner<Text,IntWritable>{

**public** **int** getPartition (Text name,IntWritable value,**int** numReduceTasks)

{

**if** (name.toString().equals("Jan"))

{

**return** 1 ;

}

**else** **if** (name.toString().equals("Feb"))

{

**return** 2;

}

**else** **if** (name.toString().equals("Mar") )

{

**return** 3;

}

**else** **if** (name.toString().equals("Apr") )

{

**return** 4; }

**else** **if** (name.toString().equals("May") )

{

**return** 5;

}

**else** **if** (name.toString().equals("Jun") )

{

**return** 6;

}

**else** **if** (name.toString().equals("Jul") )

{

**return** 7;

}

**else** **if** (name.toString().equals("Aug"))

{

**return** 8;

}

**else** **if** (name.toString().equals("Sep"))

{

**return** 9;

}

**else** **if** (name.toString().equals("Oct"))

{

**return** 10;

}

**else** **if** (name.toString().equals("Nov"))

{

**return** 11;

}

**else** **if** (name.toString().equals("Dec"))

{

**return** 12;

}

**else**{

**return** 0;

}

}

}

**public** **static** **void** main(String[] arg) **throws** Exception{

// **TODO** Auto-generated method stub

**int** res=ToolRunner.*run*(**new** Configuration(),**new** part(), arg);

System.*exit*(0);

}

@Override

**public** **int** run(String[] arg) **throws** Exception {

// **TODO** Auto-generated method stub

Configuration conf = getConf() ;

Job job = **new** Job(conf,"part");

job.setJarByClass(part.**class**);

FileInputFormat.*setInputPaths*(job,**new** Path(arg[0]));

FileOutputFormat.*setOutputPath*(job,**new** Path(arg[1]));

job.setMapperClass(Map.**class**);

job.setMapOutputKeyClass(Text.**class**) ;

job.setMapOutputValueClass(IntWritable.**class**) ;

// set partitioner statement

job.setPartitionerClass(CaderPartitioner.**class**) ;

job.setReducerClass(SumReducer.**class**);

job.setNumReduceTasks(13) ;

job.setInputFormatClass (TextInputFormat.**class**);

job.setOutputFormatClass(TextOutputFormat.**class**);

job.setOutputKeyClass(Text.**class**);

job.setOutputValueClass(Text.**class**);

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

**return** 0 ;

}

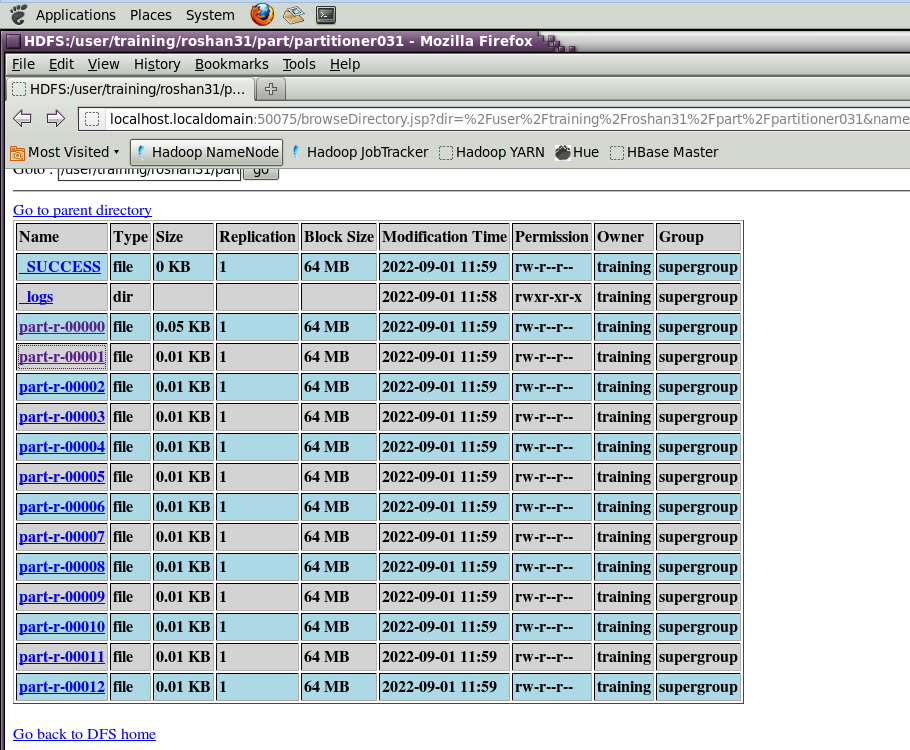
}

Text

Description automatically generated

Text, letter

Description automatically generated



Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated  
Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Even

Write a MapReduce program to find age group-wise (20 to 30,31 to 45, 46 to 60, >61) of customer count from the "cust" dataset.

Code:

**import** java.io.\*;

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

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

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

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

**import** org.apache.hadoop.mapreduce.Mapper.Context;

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

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

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

**public** **class** age **extends** Configured **implements** Tool {

**public** **static** **class** Map **extends** Mapper<LongWritable, Text, Text, Text> {

@Override **public** **void** map (LongWritable key, Text value, Context context)

**throws** IOException, InterruptedException {

String line = value.toString();

String[] cust =line.split(",");

String age=cust[3];

context.write(**new** Text (age), **new** Text(value));

}

}

**public** **static** **class** CodeReducer **extends** Reducer<Text, Text, Text, IntWritable>{

@Override

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

**int** Count = 0;

**for** (Text value: values){

Count += 1;

}

context.write(name, **new** IntWritable(Count));

}

}

**public** **static** **class** CaderPartitioner **extends** Partitioner<Text,Text>{

**public** **int** getPartition (Text name,Text value,**int** numReduceTasks)

{

String line = value.toString();

String[] cust =line.split(",");

**int** age=Integer.*parseInt*(cust[3]);

**if** (age>=20 && age<=30)

{

**return** 1 ;

}

**else** **if** (age>=31 && age<=45)

{

**return** 2;

}

**else** **if** (age>=46 && age<=60 )

{

**return** 3;

}

**else** **if** (age>=61 )

{

**return** 4;

}

**else**

{

**return** 0;

}

}

}

**public** **int** run(String[] arg) **throws** Exception {

// **TODO** Auto-generated method stub

Configuration conf = getConf() ;

Job job = **new** Job(conf,"cust");

job.setJarByClass(age.**class**);

FileInputFormat.*setInputPaths*(job,**new** Path(arg[0]));

FileOutputFormat.*setOutputPath*(job,**new** Path(arg[1]));

job.setMapperClass(Map.**class**);

job.setMapOutputKeyClass(Text.**class**) ;

job.setMapOutputValueClass(Text.**class**) ;

job.setPartitionerClass(CaderPartitioner.**class**) ;

job.setReducerClass(CodeReducer.**class**);

job.setNumReduceTasks(6) ;

job.setInputFormatClass (TextInputFormat.**class**);

job.setOutputFormatClass(TextOutputFormat.**class**);

job.setOutputKeyClass(Text.**class**);

job.setOutputValueClass(Text.**class**);

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

**return** 0 ;

}

**public** **static** **void** main(String[] arg) **throws** Exception{

// **TODO** Auto-generated method stub

**int** res=ToolRunner.*run*(**new** Configuration(),**new** age(), arg);

System.*exit*(0);

}

}

Graphical user interface, text, application, email

Description automatically generatedGraphical user interface, text, application, email

Description automatically generatedGraphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

# **Assignment : 06**

# 

# **Partitioner Demo - Cust dataset**

**Code:**

**import** java.io.\*;

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

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

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

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

**import** org.apache.hadoop.mapreduce.Mapper.Context;

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

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

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

**public** **class** cust **extends** Configured **implements** Tool{

**public** **static** **class** Map **extends** Mapper<LongWritable, Text, Text, IntWritable> {

@Override **public** **void** map (LongWritable key, Text value, Context context)

**throws** IOException, InterruptedException {

String line = value.toString();

String[] cust =line.split(",",-3);

String name=cust[4];

context.write(**new** Text (name), **new** IntWritable (1));

}

}

**public** **static** **class** CodeReducer **extends** Reducer<Text, IntWritable, Text, IntWritable>{

@Override

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

**int** Count = 0;

**for** (IntWritable value: values){

Count += value.get();

}

context.write(name, **new** IntWritable(Count));

}

}

**public** **static** **class** CaderPartitioner **extends** Partitioner<Text,IntWritable>{

**public** **int** getPartition (Text name,IntWritable value,**int** numReduceTasks)

{

**if** (name.toString().equals("Pilot"))

{

**return** 1 ;

}

**else** **if** (name.toString().equals("Teacher"))

{

**return** 2;

}

**else** **if** (name.toString().equals("Lawyer") )

{

**return** 3;

}

**else** **if** (name.toString().equals("Pharmacist") )

{

**return** 4; }

**else** **if** (name.toString().equals("Carpenter") )

{

**return** 5;

}

**else**{

**return** 0;

}

}

}

**public** **static** **void** main(String[] arg) **throws** Exception{

// **TODO** Auto-generated method stub

**int** res=ToolRunner.*run*(**new** Configuration(),**new** cust(), arg);

System.*exit*(0);

}

**public** **int** run(String[] arg) **throws** Exception {

// **TODO** Auto-generated method stub

Configuration conf = getConf() ;

Job job = **new** Job(conf,"cust");

job.setJarByClass(cust.**class**);

FileInputFormat.*setInputPaths*(job,**new** Path(arg[0]));

FileOutputFormat.*setOutputPath*(job,**new** Path(arg[1]));

job.setMapperClass(Map.**class**);

job.setMapOutputKeyClass(Text.**class**) ;

job.setMapOutputValueClass(IntWritable.**class**) ;

// set partitioner statement

job.setPartitionerClass(CaderPartitioner.**class**) ;

job.setReducerClass(CodeReducer.**class**);

job.setNumReduceTasks(6) ;

job.setInputFormatClass (TextInputFormat.**class**);

job.setOutputFormatClass(TextOutputFormat.**class**);

job.setOutputKeyClass(Text.**class**);

job.setOutputValueClass(Text.**class**);

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

**return** 0 ;

}

}

Graphical user interface, text, application, email

Description automatically generated

Text

Description automatically generated

Table

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

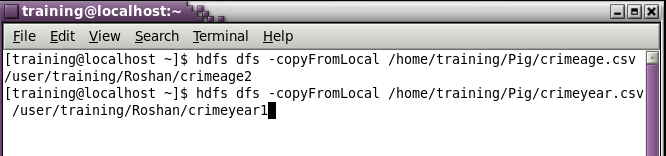
Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

# **Assignment : 07**

**Government Crime Assignment - Pig**





## **1.To find the total number of crimes which occurred in all the states in the year 2006.**

crimeyear7 = load '/user/training/Roshan/crimeyear2' using PigStorage(',') AS (STATE:chararray, CRIMEHEAD:chararray, one:int, two:int, three:int, four:int, five:int, six:int, seven:int, eight:int, nine:int, ten:int, eleven:int, twelve:int);

G = GROUP crimeyear7 BY STATE;

C = FOREACH G generate group, SUM(crimeyear7.six);

Dump C;



**2. Calculate the number of females (age between 18-30 years) who were victims in different crimes in different states**

crimeage = LOAD '/user/training/Roshan/crimeage' using PigStorage(',') AS (STATE1:chararray, Crimehead1:chararray, Male\_Below\_18\_Years:int, Female\_Below\_18\_Years:int, Male\_Between\_18\_30\_Years:int, Female\_Between\_18\_30\_Years:int, Male\_Between\_30\_45\_Years:int, Female\_Between\_30\_45\_Years:int, Male\_Between\_45\_60\_Years:int, Female\_Between\_45\_60\_Years:int, Male\_Above\_60\_Years:int, Female\_Above\_60\_Years:int, Total\_Male:int, Total\_Female:int, Grand\_Total:int);

G = GROUP crimeage ALL;

C = FOREACH G generate group, SUM(crimeage.Female\_Between\_18\_30\_Years);



## **3. Which state have highest rape cases across all years.**

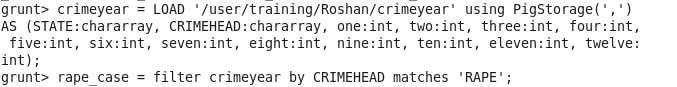
crimeyear = LOAD '/user/training/Roshan/crimeyear' using PigStorage(',') AS (STATE:chararray, CRIMEHEAD:chararray, one:int, two:int, three:int, four:int, five:int, six:int, seven:int, eight:int, nine:int, ten:int, eleven:int, twelve:int);

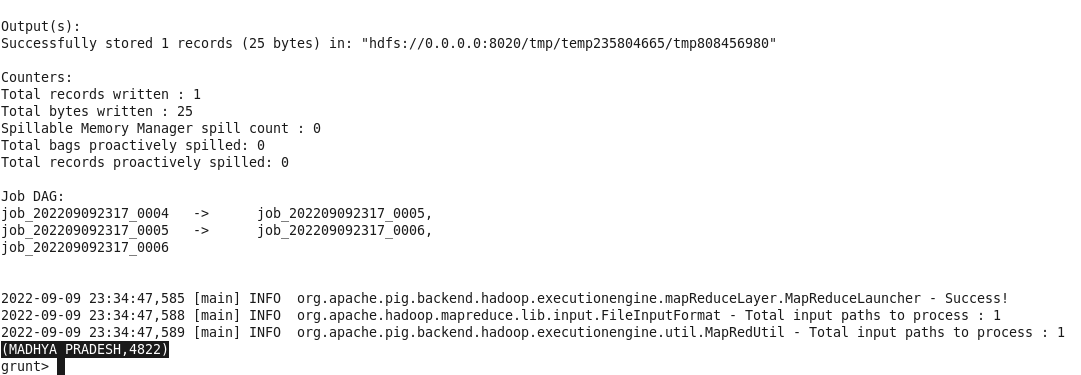
rape\_case = filter crimeyear by CRIMEHEAD matches 'RAPE';

g = FOREACH rape\_case generate $0,$14;

c = ORDER g by $1 desc;

r = LIMIT c 1;





**4. Find crime wise which state has the highest count.**

crimeage = LOAD '/user/training/Roshan/crimeage' using PigStorage(',') AS (STATE1:chararray, Crimehead1:chararray, Male\_Below\_18\_Years:int, Female\_Below\_18\_Years:int, Male\_Between\_18\_30\_Years:int, Female\_Between\_18\_30\_Years:int, Male\_Between\_30\_45\_Years:int, Female\_Between\_30\_45\_Years:int, Male\_Between\_45\_60\_Years:int, Female\_Between\_45\_60\_Years:int, Male\_Above\_60\_Years:int, Female\_Above\_60\_Years:int, Total\_Male:int, Total\_Female:int, Grand\_Total:int);

count = FOREACH crimeage generate $0, $1, $14;

count2 = group count all;

max = foreach count2 generate MAX(count.$2) as cr;

s\_count = filter count by $2 == max.cr;

