Module 4 – Advance MapReduce

Assignment – Practice “Advance MapReduce” present in the LMS.

1. **Combiner and Partitioner**

Code:

**package** in.edureka.mapreduce;

**import** java.io.IOException;

**import** java.util.Iterator;

**import** java.util.StringTokenizer;

**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.mapred.FileInputFormat;

**import** org.apache.hadoop.mapred.FileOutputFormat;

**import** org.apache.hadoop.mapred.JobClient;

**import** org.apache.hadoop.mapred.JobConf;

**import** org.apache.hadoop.mapred.MapReduceBase;

**import** org.apache.hadoop.mapred.Mapper;

**import** org.apache.hadoop.mapred.OutputCollector;

**import** org.apache.hadoop.mapred.Partitioner;

**import** org.apache.hadoop.mapred.Reducer;

**import** org.apache.hadoop.mapred.Reporter;

**import** org.apache.hadoop.mapred.TextInputFormat;

**import** org.apache.hadoop.mapred.TextOutputFormat;

**public** **class** WithCombiner\_and\_Partitioner {

**public** **static** **class** Map **extends** MapReduceBase **implements**

Mapper<LongWritable, Text, Text, IntWritable> {

@Override

**public** **void** map(LongWritable key, Text value,

OutputCollector<Text, IntWritable> output, Reporter reporter)

**throws** IOException {

String line = value.toString();

StringTokenizer tokenizer = **new** StringTokenizer(line);

**while** (tokenizer.hasMoreTokens()) {

value.set(tokenizer.nextToken());

output.collect(value, **new** IntWritable(1));

// // I am fine I am fine

// v

// I 1

// am 1

// fine 1

// I 1

// am 1

// fine 1

// I (1,1)

}

}

}

// Output types of Mapper should be same as arguments of Partitioner

**public** **static** **class** MyPartitioner **implements** Partitioner<Text, IntWritable> {

@Override

**public** **int** getPartition(Text key, IntWritable value, **int** numPartitions) {

String myKey = key.toString().toLowerCase();

**if** (myKey.equals("hadoop")) {

**return** 0;

}

**if** (myKey.equals("data")) {

**return** 1;

} **else** {

**return** 2;

}

}

@Override

**public** **void** configure(JobConf arg0) {

// Gives you a new instance of JobConf if you want to change Job

// Configurations

}

}

**public** **static** **class** Reduce **extends** MapReduceBase **implements**

Reducer<Text, IntWritable, Text, IntWritable> {

@Override

**public** **void** reduce(Text key, Iterator<IntWritable> values,

OutputCollector<Text, IntWritable> output, Reporter reporter)

**throws** IOException {

**int** sum = 0;

**while** (values.hasNext()) {

sum += values.next().get();

// sum = sum + 1;

}

// beer,3

output.collect(key, **new** IntWritable(sum));

}

}

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

JobConf conf = **new** JobConf(WithCombiner\_and\_Partitioner.**class**);

conf.setJobName("wordcount");

// Forcing program to run 3 reducers

conf.setNumReduceTasks(3);

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

conf.setCombinerClass(Reduce.**class**);

conf.setReducerClass(Reduce.**class**);

conf.setPartitionerClass(MyPartitioner.**class**);

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

conf.setOutputValueClass(IntWritable.**class**);

conf.setInputFormat(TextInputFormat.**class**);

conf.setOutputFormat(TextOutputFormat.**class**);

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

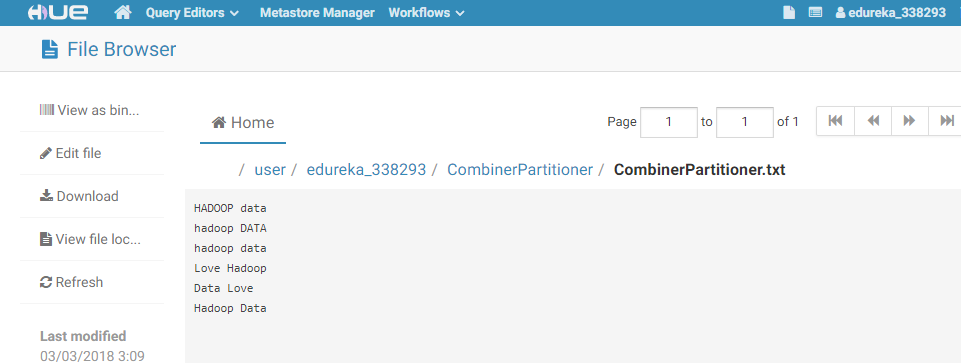
FileOutputFormat.*setOutputPath*(conf, **new** Path(args[1]));

JobClient.*runJob*(conf);

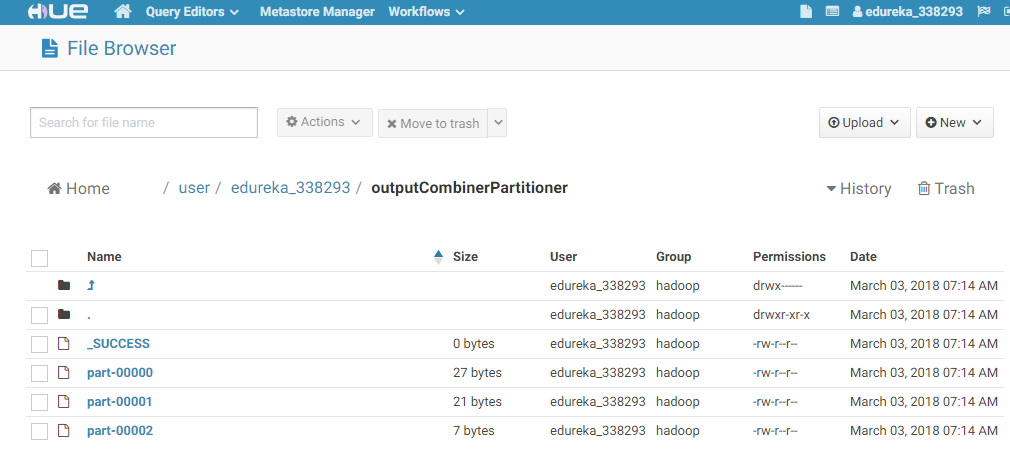
}

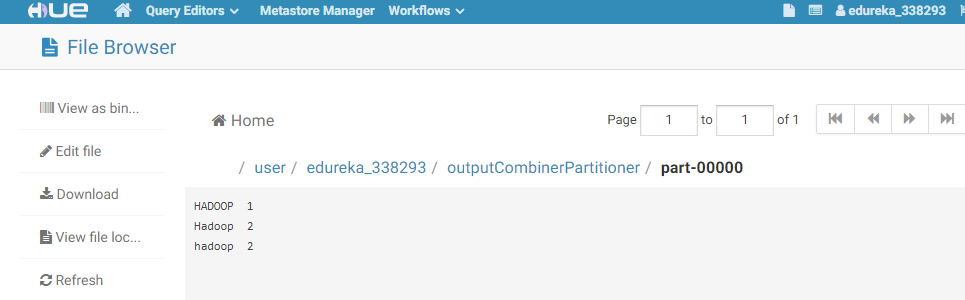
}

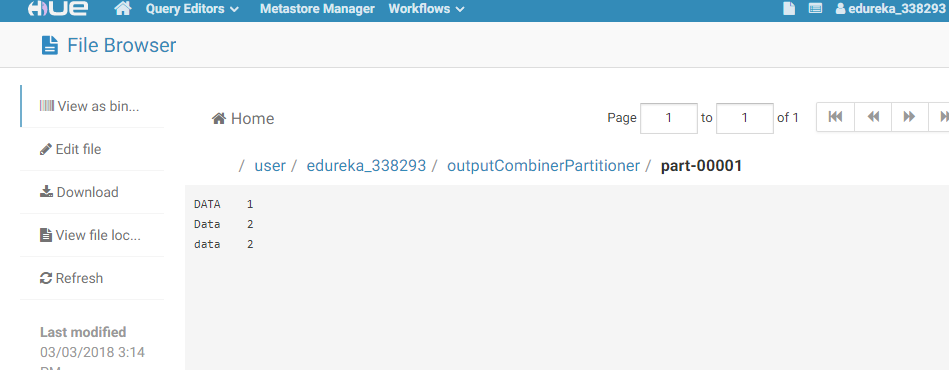
Input:

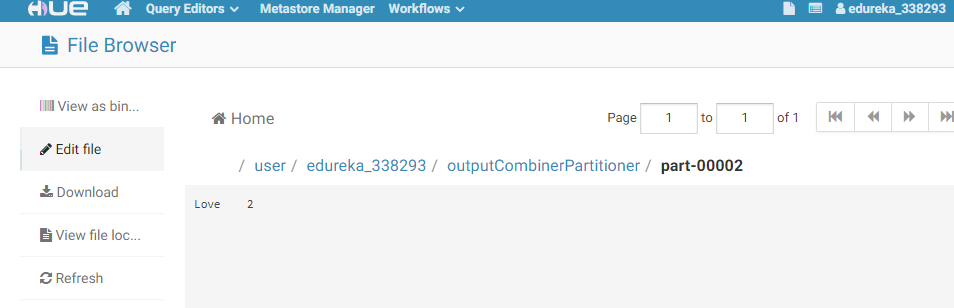


Output:









1. **My Counter**

Code:

**package** in.edureka.mapreduce;

**import** java.io.IOException;

**import** java.util.Date;

**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.Mapper.Context;

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

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

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

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

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

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

**public** **class** MyCounter {

**public** **static** **enum** MONTH{

***DEC***,

***JAN***,

***FEB***

};

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

**private** Text out = **new** Text();

**protected** **void** map(LongWritable key, Text value, Context context)

**throws** java.io.IOException, InterruptedException {

String line = value.toString();

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

**long** lts = Long.*parseLong*(strts[1]);

Date time = **new** Date(lts);

**int** m = time.~~getMonth~~();

**if**(m==11){

context.getCounter(MONTH.***DEC***).increment(10);

}

**if**(m==0){

context.getCounter(MONTH.***JAN***).increment(20);

}

**if**(m==1){

context.getCounter(MONTH.***FEB***).increment(30);

}

out.set("success");

context.write(out,out);

}

}

**public** **static** **void** main(String[] args)

**throws** IOException, ClassNotFoundException, InterruptedException {

Job job = **new** ~~Job~~();

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

job.setJobName("CounterTest");

job.setNumReduceTasks(0);

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

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

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

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

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

job.waitForCompletion(**true**);

Counters counters = job.getCounters();

Counter c1 = counters.findCounter(MONTH.***DEC***);

System.***out***.println(c1.getDisplayName()+ " : " + c1.getValue());

c1 = counters.findCounter(MONTH.***JAN***);

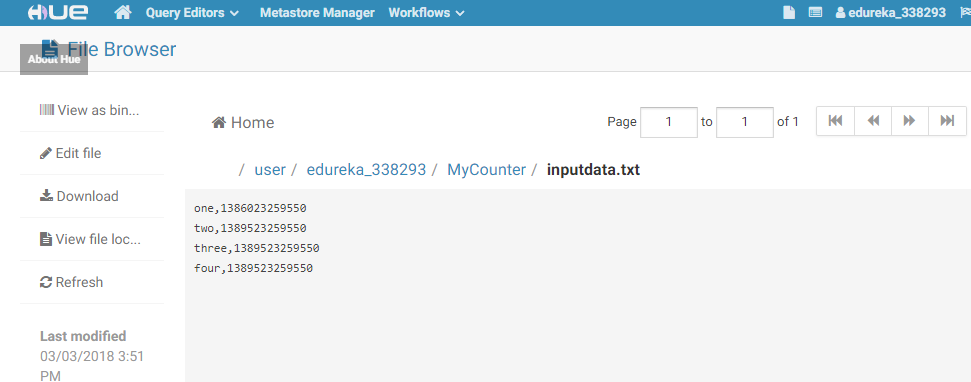
System.***out***.println(c1.getDisplayName()+ " : " + c1.getValue());

c1 = counters.findCounter(MONTH.***FEB***);

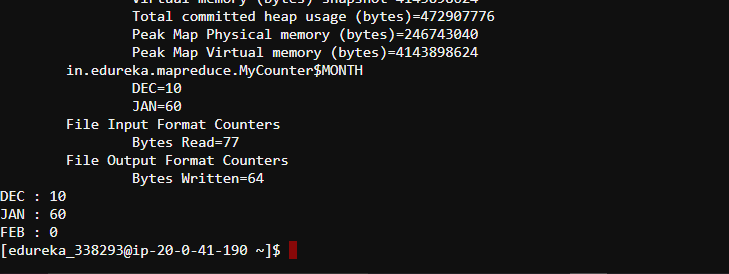
System.***out***.println(c1.getDisplayName()+ " : " + c1.getValue());

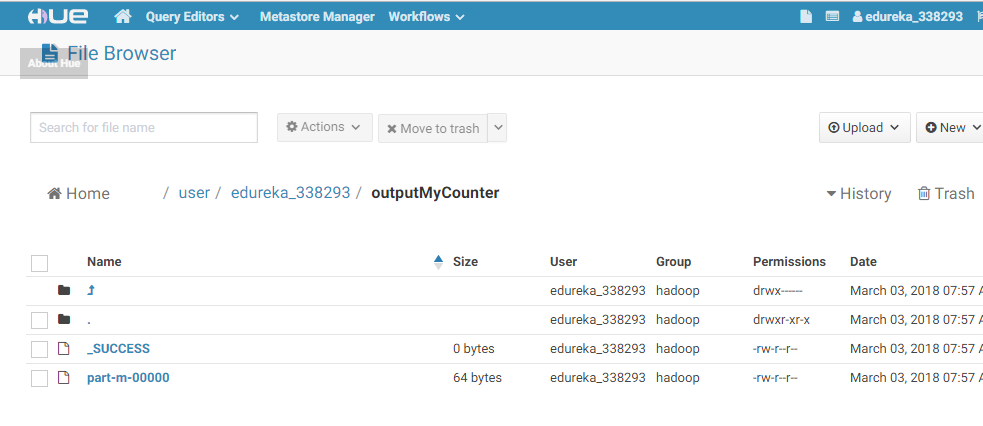
}

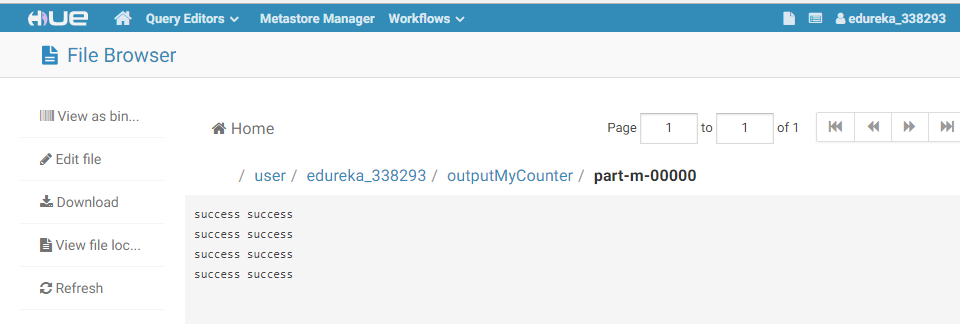
}

Input:

Output:







1. **Distributed Cache**

Code:

**package** in.edureka.mapreduce;

**import** java.io.BufferedReader;

**import** java.io.FileReader;

**import** java.io.IOException;

**import** java.net.URI;

**import** java.util.HashMap;

**import** java.util.Map;

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

**import** org.apache.hadoop.filecache.DistributedCache;

**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.Mapper.Context;

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

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

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

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

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

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

**public** **class** MyDC {

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

**private** Map<String, String> abMap = **new** HashMap<String, String>();

**private** Text outputKey = **new** Text();

**private** Text outputValue = **new** Text();

**protected** **void** setup(Context context) **throws** java.io.IOException, InterruptedException{

Path[] files = DistributedCache.getLocalCacheFiles(context.getConfiguration());

**for** (Path p : files) {

**if** (p.getName().equals("abc.dat")) {

BufferedReader reader = **new** BufferedReader(**new** FileReader(p.toString()));

String line = reader.readLine();

**while**(line != **null**) {

String[] tokens = line.split("\t");

String ab = tokens[0];

String state = tokens[1];

abMap.put(ab, state);

line = reader.readLine();

}

}

}

**if** (abMap.isEmpty()) {

**throw** **new** IOException("Unable to load Abbrevation data.");

}

}

**protected** **void** map(LongWritable key, Text value, Context context)

**throws** java.io.IOException, InterruptedException {

String row = value.toString();

String[] tokens = row.split("\t");

String inab = tokens[0];

String state = abMap.get(inab);

outputKey.set(state);

outputValue.set(row);

context.write(outputKey,outputValue);

}

}

**public** **static** **void** main(String[] args)

**throws** IOException, ClassNotFoundException, InterruptedException {

Job job = **new** Job();

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

job.setJobName("DCTest");

job.setNumReduceTasks(0);

**try**{

DistributedCache.addCacheFile(**new** URI("abc.dat"), job.getConfiguration());

}**catch**(Exception e){

System.out.println(e);

}

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

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

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

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

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

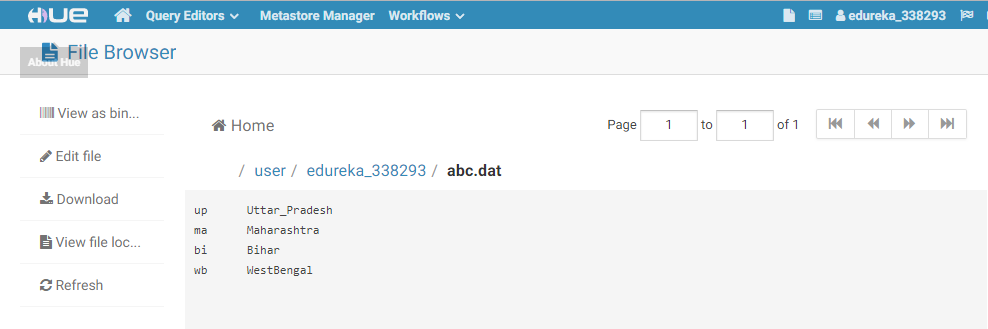
job.waitForCompletion(**true**);

}

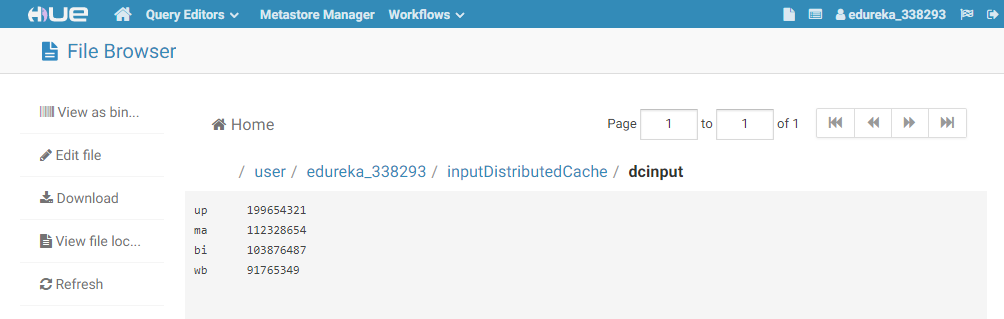
}

Input:

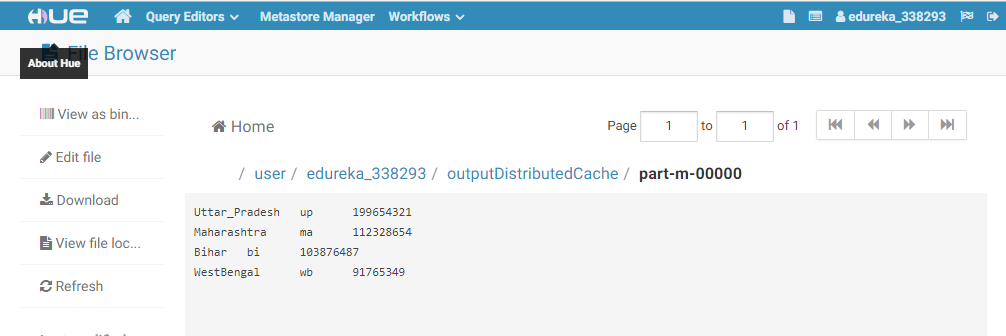
abc.dat file



Dcinput file



Output:



1. **Reduce Join**

Code:

**package** in.edureka.mapreduce;

**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.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;

**public** **class** ReduceJoin {

**public** **static** **class** CustsMapper **extends**

Mapper<Object, Text, Text, Text> {

**public** **void** map(Object key, Text value, Context context)

**throws** IOException, InterruptedException {

String record = value.toString();

String[] parts = record.split(",");

context.write(**new** Text(parts[0]), **new** Text("custs\t" + parts[1]));

}

}

**public** **static** **class** TxnsMapper **extends**

Mapper<Object, Text, Text, Text> {

**public** **void** map(Object key, Text value, Context context)

**throws** IOException, InterruptedException {

String record = value.toString();

String[] parts = record.split(",");

context.write(**new** Text(parts[2]), **new** Text("txns\t" + parts[3]));

}

}

**public** **static** **class** ReduceJoinReducer **extends**

Reducer<Text, Text, Text, Text> {

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

**throws** IOException, InterruptedException {

String name = "";

**double** total = 0.0;

**int** count = 0;

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

String parts[] = t.toString().split("\t");

**if** (parts[0].equals("txns")) {

count++;

total += Float.*parseFloat*(parts[1]);

} **else** **if** (parts[0].equals("custs")) {

name = parts[1];

}

}

String str = String.*format*("%d\t%f", count, total);

context.write(**new** Text(name), **new** Text(str));

}

}

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

Configuration conf = **new** Configuration();

Job job = **new** ~~Job~~(conf, "Reduce-side join");

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

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

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

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

MultipleInputs.*addInputPath*(job, **new** Path(args[0]),TextInputFormat.**class**, CustsMapper.**class**);

MultipleInputs.*addInputPath*(job, **new** Path(args[1]),TextInputFormat.**class**, TxnsMapper.**class**);

Path outputPath = **new** Path(args[2]);

FileOutputFormat.*setOutputPath*(job, outputPath);

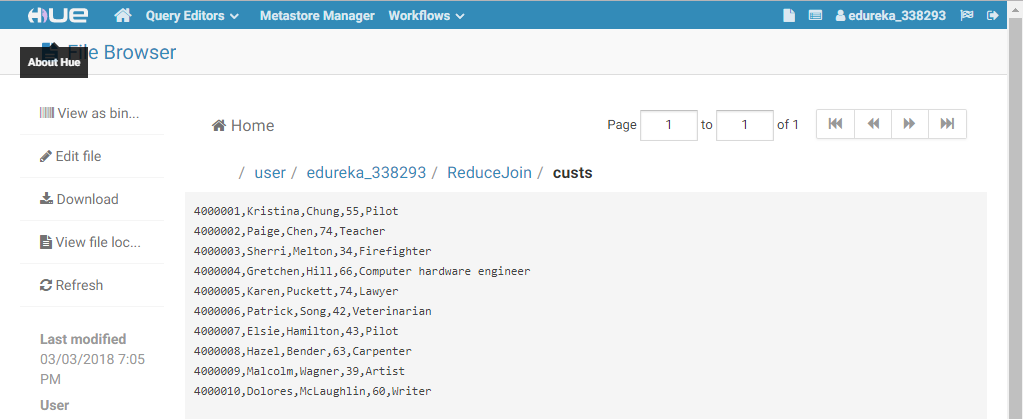
outputPath.getFileSystem(conf).~~delete~~(outputPath);

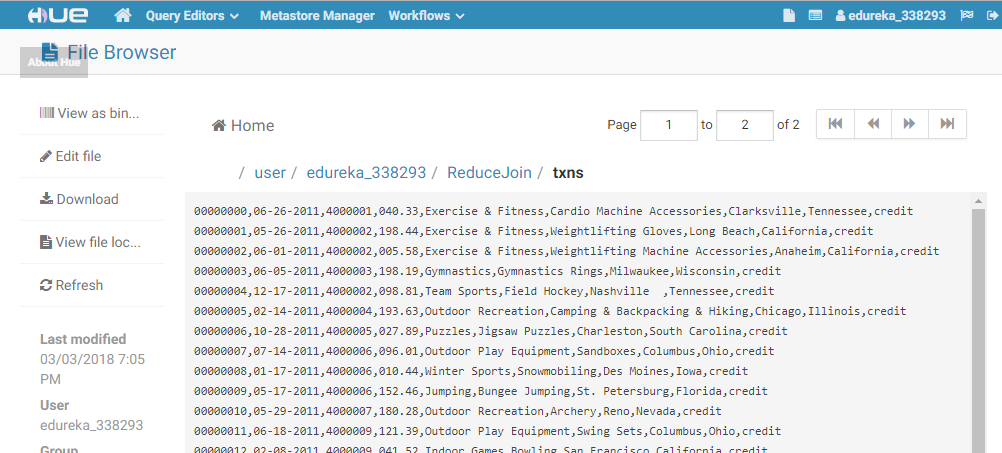
System.*exit*(job.waitForCompletion(**true**) ? 0 : 1);

}

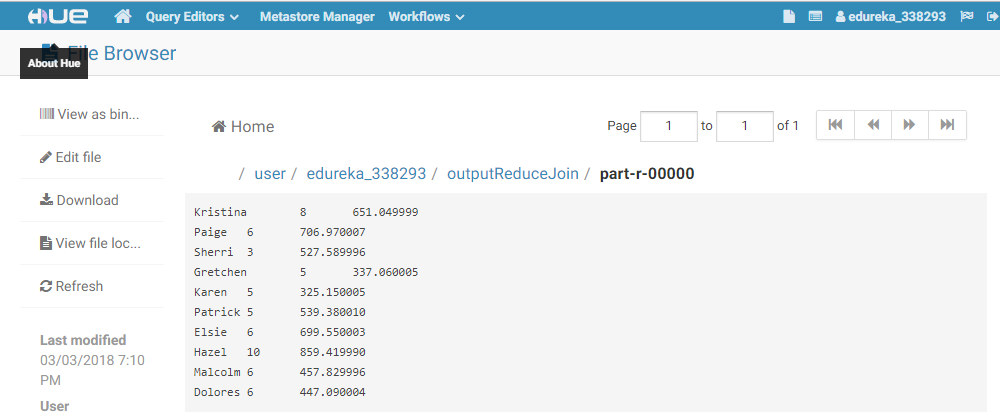
}

Input:





Output:



1. **Sequence File**

Code:

Converting into Binary File:

**package** in.edureka.mapreduce;

import java.io.ByteArrayOutputStream;

import java.io.IOException;

import java.net.URI;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.FSDataInputStream;

import org.apache.hadoop.fs.FileSystem;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.BytesWritable;

import org.apache.hadoop.io.IOUtils;

import org.apache.hadoop.io.Text;

import org.apache.hadoop.mapreduce.Job;

import org.apache.hadoop.mapreduce.Mapper;

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

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

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

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

import org.apache.hadoop.util.GenericOptionsParser;

import org.apache.commons.\*;

import org.apache.commons.logging.impl.Log4JLogger;

public class BinaryFilesToHadoopSequenceFile {

//Log4JLogger logger = new Log4JLogger().getLogger();

public static class BinaryFilesToHadoopSequenceFileMapper extends Mapper<Object, Text, Text, BytesWritable> {

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

throws IOException, InterruptedException {

//logger.info("map method called..");

String uri = value.toString();

Configuration conf = new Configuration();

FileSystem fs = FileSystem.get(URI.create(uri), conf);

FSDataInputStream in = null;

try {

in = fs.open(new Path(uri));

java.io.ByteArrayOutputStream bout = new ByteArrayOutputStream();

byte buffer[] = new byte[1024 \* 1024];

while( in.read(buffer, 0, buffer.length) >= 0 ) {

bout.write(buffer);

}

context.write(value, new BytesWritable(bout.toByteArray()));

} finally {

IOUtils.closeStream(in);

}

}

}

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: BinaryFilesToHadoopSequenceFile <in Path for url file> <out pat for sequence file>");

// System.exit(2);

// }

Job job = new Job(conf, "BinaryFilesToHadoopSequenceFile");

job.setJarByClass(BinaryFilesToHadoopSequenceFile.class);

job.setMapperClass(BinaryFilesToHadoopSequenceFileMapper.class);

job.setOutputKeyClass(Text.class);

job.setOutputValueClass(BytesWritable.class);

job.setInputFormatClass(TextInputFormat.class);

job.setOutputFormatClass(SequenceFileOutputFormat.class);

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

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

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

}

}

Duplicate image code:

**package** in.edureka.mapreduce;

import java.io.IOException;

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

import org.apache.hadoop.conf.Configuration;

import org.apache.hadoop.fs.Path;

import org.apache.hadoop.io.BytesWritable;

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.input.SequenceFileInputFormat;

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

import org.apache.hadoop.util.GenericOptionsParser;

public class ImageDuplicatesMapper extends Mapper<Text, BytesWritable, Text, Text>{

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

//get the md5 for this specific file

String md5Str;

try {

md5Str = calculateMd5(value.getBytes());

} catch (NoSuchAlgorithmException e) {

e.printStackTrace();

context.setStatus("Internal error - can't find the algorithm for calculating the md5");

return;

}

Text md5Text = new Text(md5Str);

//put the file in the map where the md5 is the key, so duplicates will

// be grouped together for the reduce function

context.write(md5Text, key);

}

static String calculateMd5(byte[] imageData) throws NoSuchAlgorithmException {

//get the md5 for this specific data

MessageDigest md = MessageDigest.getInstance("MD5");

md.update(imageData);

byte[] hash = md.digest();

// Below code of converting Byte Array to hex

String hexString = new String();

for (int i=0; i < hash.length; i++) {

hexString += Integer.toString( ( hash[i] & 0xff ) + 0x100, 16).substring( 1 );

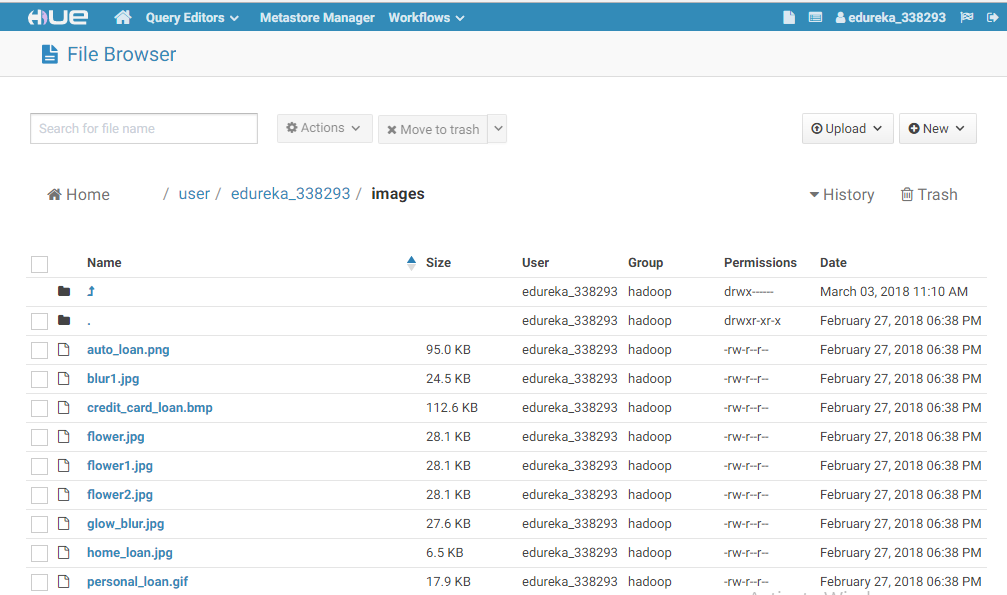
}

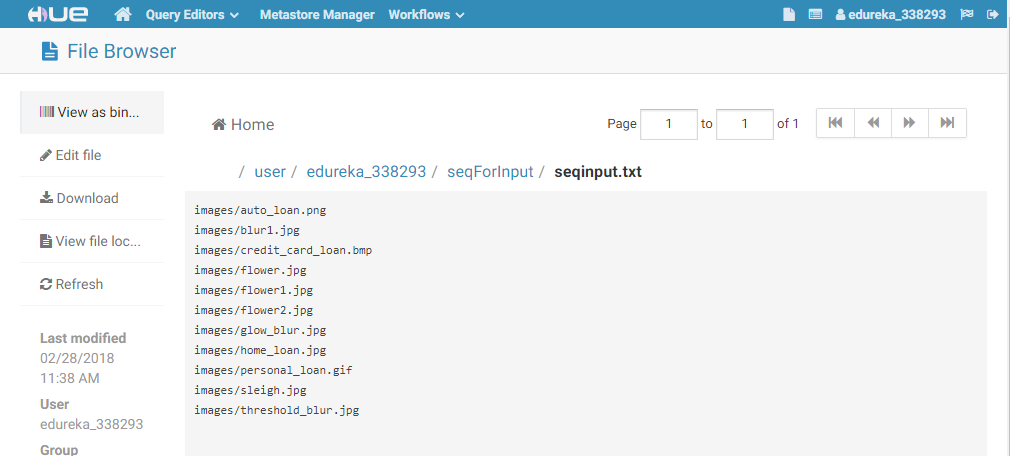
return hexString;

}

}

Input:





Output:

