LAB 9 HADOOP MAPREDUCE PROGRAM THE USE OF JOIN:-

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) {}

@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 Name input'");

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:-

```
// 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€;
```

```
}

}

}

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€;

}

}

@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);

}
```

}

}

OUTPUT:-

```
c:\hadoop_new\share\hadoop\mapreduce>hdfs dfs -cat \joinOutput\part-00000
"100005361"     "2"             "36134"
"100018705"     "2"             "76"
"100022094"     "0"             "6354"
```