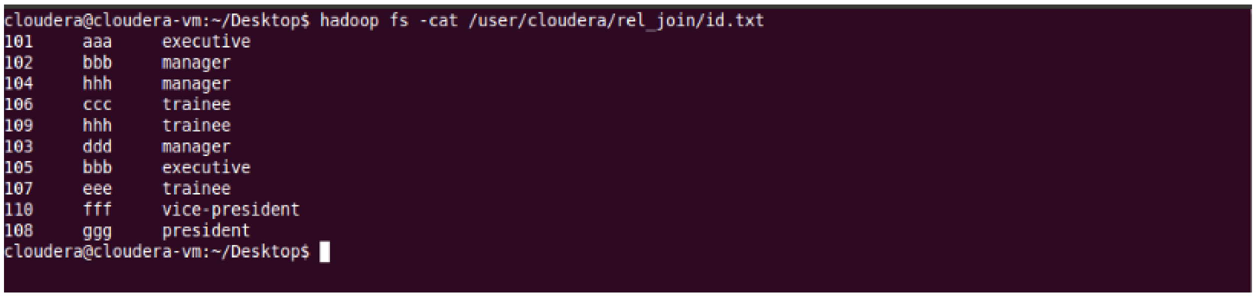
Assignment5

In this assignment of **Relational Join,** we will join two tables. These two tables are present in two files named *id.txt* and *trips.txt*.

<u>id.txt</u>

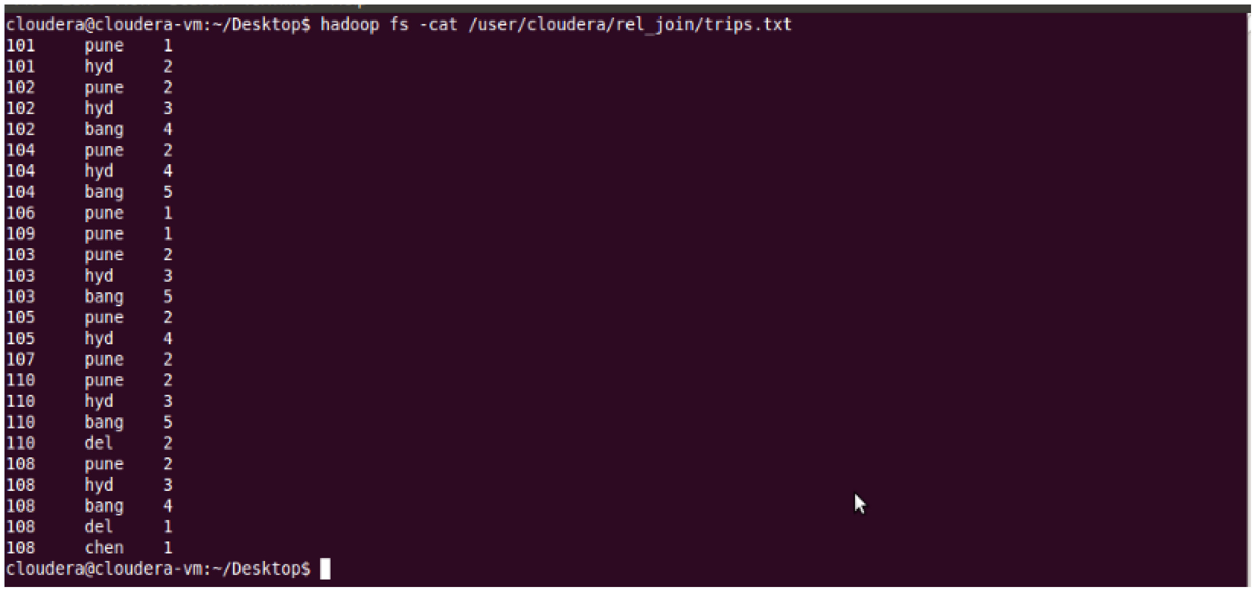
This file has id wise data for employees company.



```
cloudera@cloudera-vm:~/Desktop$ hadoop fs -cat /user/cloudera/rel_join/id.txt
101     aaa     executive
102     bbb     manager
104     hhh     manager
106     ccc     trainee
109     hhh     trainee
103     ddd     manager
105     bbb     executive
107     eee     trainee
110     fff     vice-president
108     ggg     president
cloudera@cloudera-vm:~/Desktop$ 
```

<u>trips.txt</u>

This file has trip details for each employee.



```
cloudera@cloudera-vm:~/Desktop$ hadoop fs -cat /user/cloudera/rel_join/trips.txt
101     pune    1
101     hyd     2
102     pune    2
102     hyd     3
102     bang    4
104     pune    2
104     hyd     4
104     bang    5
106     pune    1
109     pune    1
103     pune    2
103     hyd     3
103     bang    5
105     pune    2
105     hyd     4
107     pune    2
110     pune    2
110     hyd     3
110     bang    5
110     del     2
108     pune    2
108     hyd     3
108     bang    4
108     del     1
108     chen    1
cloudera@cloudera-vm:~/Desktop$ 
```

Objective: We have to join these two data files based on employee id.

<u>Solution:</u>

In this case, we will be giving input to our map reduce program in the form of two files. Certain new commands will be introduced in the *run.class.* Also we will be writing two map classes one for each file to keep the track of our data.

Mapper class for "id.txt"(it's a tab delimited file)

```
importjava.io.IOException;
importjava.util.StringTokenizer;

importorg.apache.hadoop.io.DoubleWritable;
importorg.apache.hadoop.io.LongWritable;
importorg.apache.hadoop.io.Text;
importorg.apache.hadoop.mapred.MapReduceBase;
importorg.apache.hadoop.mapred.Mapper;
importorg.apache.hadoop.mapred.OutputCollector;
importorg.apache.hadoop.mapred.Reporter;

public class map_id extends MapReduceBase implements
Mapper<LongWritable,Text,DoubleWritable,Text> {
        private final String id_file = "zzzzzzzzzz";
        public void map(LongWritable key, Text value,
OutputCollector<DoubleWritable,Text>output,Reporter reporter) throws IOException {
                String line=value.toString();
                String outline=id_file;
                double id=0;
                StringTokenizer token = new StringTokenizer(line);
                if ( token.hasMoreTokens()){
                        id = Double.parseDouble(token.nextToken());
                }
                while (token.hasMoreTokens() ){
                        outline = outline + '\t' +token.nextToken();
                }
                output.collect(new DoubleWritable(id),new Text(outline));
        }
}
```

Counter added to identify data from "id.txt"

First token is converted to double as employee id which will serve as a key to be sent to reducer program.

1) Remaining tokens are added to token "zzzzzzzzz" for identification during reduce phase.
   Reason for adding our 'identification' token.
   Output from our mapper function will be sent to reducer function. So all the values with same key from both files will be sent to reducer function. We have added out token so that we can identify values from "id.txt" so that these values can be added to values from "trips.txt" and hence final joined values can be sent to output collector.

Mapper class for "trips.txt"

```
importjava.io.IOException;
importjava.util.StringTokenizer;

importorg.apache.hadoop.io.DoubleWritable;
importorg.apache.hadoop.io.LongWritable;
importorg.apache.hadoop.io.Text;
importorg.apache.hadoop.mapred.MapReduceBase;
importorg.apache.hadoop.mapred.Mapper;
importorg.apache.hadoop.mapred.OutputCollector;
importorg.apache.hadoop.mapred.Reporter;


public class map_trips extends MapReduceBase implements
Mapper<LongWritable,Text,DoubleWritable,Text> {

        public void map(LongWritable key, Text value,
OutputCollector<DoubleWritable,Text>output,Reporter reporter) throws IOException {
                String line=value.toString();
                String outline2=null;
                int counter =0;
                double id=0;
                StringTokenizer token = new StringTokenizer(line);

                if ( token.hasMoreTokens()){
                        id = Double.parseDouble(token.nextToken());

                }
                while (token.hasMoreTokens() ){
                        if(counter == 1)
                                outline2 = outline2 + ('\t'+token.nextToken());
                        else{
                                outline2 = token.nextToken();
                                counter =1;
                        }
                }
                output.collect(new DoubleWritable(id),new Text(outline2));
        }
}
```

1) First token is converted to double as employee id.
2) Remaining tokens are added together to form a string and sent as value to reduce phase.

. 'counter' keeps count if we have taken out second token or not. Once second token is taken out,through counter remaining tokens are added to previous string with tab space between them.

Reducer Class

```
importjava.io.IOException;
importjava.util.ArrayList;
importjava.util.Iterator;
importjava.util.StringTokenizer;

importorg.apache.hadoop.io.DoubleWritable;
importorg.apache.hadoop.io.Text;
importorg.apache.hadoop.mapred.MapReduceBase;
importorg.apache.hadoop.mapred.OutputCollector;
importorg.apache.hadoop.mapred.Reducer;
importorg.apache.hadoop.mapred.Reporter;

public class reduce extends MapReduceBase implements Reducer<DoubleWritable,Text,DoubleWritable,Text> {
        public void reduce (DoubleWritable key, Iterator<Text>values,OutputCollector<DoubleWritable,Text>
output, Reporter reporter) throws IOException {

                ArrayList<String>arr = new ArrayList<String>();
                arr.clear();
                String line = null;
                String base=null;
                int counter = 0;
                int first=0;
                int i=0;

                while(values.hasNext())     {
                        line=values.next().toString();
                        StringTokenizer token = new StringTokenizer(line);
                        if(token.countTokens()==3){
                                token.nextToken();
                                while (token.hasMoreTokens()){
                                        if(first==0){
                                                base=token.nextToken();
                                                first=1;}
                                        else
                                                base=base+'\t'+token.nextToken();
                                        }
                                }
                        else {
                                arr.add(line);
                                counter++;}
                        }
                while(i<(counter)){
                        line = base +" "+'\t'+arr.get(i);
                        output.collect(key, new Text(line));
                        i++;}}}
```

1) If first value after tokenizing, has 3 tokens (due to our added token “zzzzzz”, it is identified as value from id.txt.
2) Then first added token is taken out and rest are joined to form base string.
3) If number of tokens are not 3,(value from trips.txt) the string is sent into an array list for future reference.

First token from “id.txt” is not included because its our added token for identification. ‘first’ keeps count if we have taken out second token or not. Once second token is taken out,through first remaining tokens are added to previous string with tab space between them.

Finally all the strings from array list are first added to base string to be sent as value to output collector with key as employee id giving the final required table.

Runner Class

```
importjava.io.IOException;

importorg.apache.hadoop.fs.Path;
importorg.apache.hadoop.io.DoubleWritable;
importorg.apache.hadoop.io.Text;
importorg.apache.hadoop.mapred.FileOutputFormat;
importorg.apache.hadoop.mapred.JobClient;
importorg.apache.hadoop.mapred.JobConf;
importorg.apache.hadoop.mapred.TextInputFormat;
importorg.apache.hadoop.mapred.TextOutputFormat;
importorg.apache.hadoop.mapred.lib.MultipleInputs;
importorg.apache.hadoop.mapred.FileInputFormat;



public class run {
        public static void main(String[] args) throws IOException{
                JobConfconf = new JobConf(run.class);
                conf.setJobName("Relational Join");

                conf.setOutputKeyClass(DoubleWritable.class);
                conf.setOutputValueClass(Text.class);



                conf.setReducerClass(reduce.class);


                conf.setOutputFormat(TextOutputFormat.class);


                MultipleInputs.addInputPath(conf, new Path(args[0]), TextInputFormat.class, map_id.class);
                MultipleInputs.addInputPath(conf, new Path(args[1]), TextInputFormat.class, map_trips.class);
                FileOutputFormat.setOutputPath(conf,new Path(args[2]));


                JobClient.runJob(conf);

        }

}
```

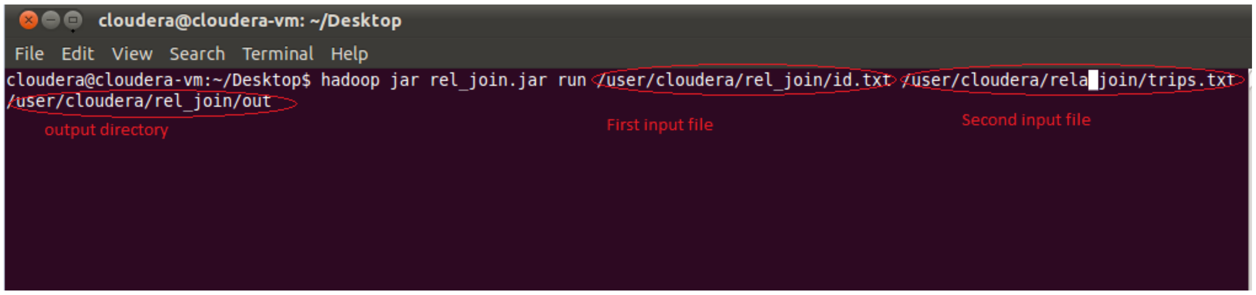
<ins>Runner Class</ins>

1) Initialise a JobConf class mentioning runner class in the constructor.
2) Set Output Key and Output Value Class. In our case we have output key as double(employee id) and output value as Text(employee's name, designation, place and roundtrips).
3) Reducer class is set. Mapper class is set individually for both inputs.
4) Output Format are set. In our case our input data and output result both are in Text Format. MultipleInputs class adds multiple inputs with different mapper classes.
5) Paths are mentioned through command line arguments.

MultipleInputs class to add multiple inputs to put map reduce program with arguments as (Jobconf,path,InputFormatclass,mapper class)

Running the program :

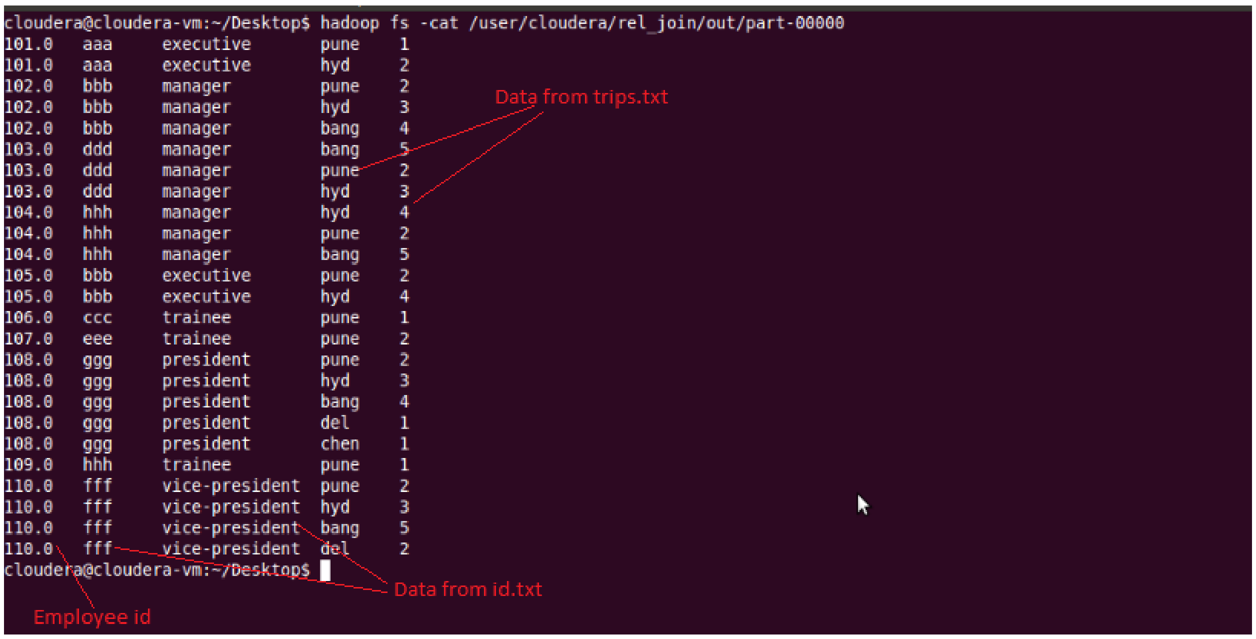
1) Make sure both files "id.txt" and "trips.txt" are present in your hdfs system.
2) Run the following command .



```
cloudera@cloudera-vm:~/Desktop$ hadoop jar rel_join.jar run /user/cloudera/rel_join/id.txt /user/cloudera/rel_join/trips.txt
/user/cloudera/rel_join/out
```

cloudera@cloudera-vm: ~/Desktop
File Edit View Search Terminal Help
output directory
First input file
Second input file

3) Final output



```
cloudera@cloudera-vm:~/Desktop$ hadoop fs -cat /user/cloudera/rel_join/out/part-00000
101.0   aaa     executive       pune    1
101.0   aaa     executive       hyd     2
102.0   bbb     manager         pune    2
102.0   bbb     manager         hyd     3
102.0   bbb     manager         bang    4
103.0   ddd     manager         bang    5
103.0   ddd     manager         pune    2
103.0   ddd     manager         hyd     3
104.0   hhh     manager         hyd     4
104.0   hhh     manager         pune    2
104.0   hhh     manager         bang    5
105.0   bbb     executive       pune    2
105.0   bbb     executive       hyd     4
106.0   ccc     trainee         pune    1
107.0   eee     trainee         pune    2
108.0   ggg     president       pune    2
108.0   ggg     president       hyd     3
108.0   ggg     president       bang    4
108.0   ggg     president       del     1
108.0   ggg     president       chen    1
109.0   hhh     trainee         pune    1
110.0   fff     vice-president  pune    2
110.0   fff     vice-president  hyd     3
110.0   fff     vice-president  bang    5
110.0   fff     vice-president  del     2
cloudera@cloudera-vm:~/Desktop$
```