```
package org.myorg;

import java.io.*;
import java.util.*;

import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.filecache.DistributedCache;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.util.*;

public class Query_3 extends Configured implements Tool {

public static class Map_1 extends MapReduceBase implements Mapper<LongWritable, Text, IntWritable, Text> {
      private IntWritable custID = new IntWritable();
      private Text transValue = new Text();

      public void map(LongWritable key, Text value, OutputCollector<IntWritable, Text> output, Reporter reporter) throws IOException {
        String line = value.toString();

        StringTokenizer tokenizer = new StringTokenizer(line, ",");
        tokenizer.nextToken();
        //assign the value of the second attribute, CustID, to custID
        custID.set(Integer.parseInt(tokenizer.nextToken()));
        //assign the value of the third and fourth attributes, TransTotal and TransNumItems, to transValue
        transValue.set("1,"+tokenizer.nextToken() +"," + tokenizer.nextToken());
          output.collect(custID, transValue);  //adds the key-value pair to the output
        }

      }

    public static class Reduce_1 extends MapReduceBase implements Reducer<IntWritable, Text, IntWritable, Text> {
      Text transValue = new Text();

        public void reduce(IntWritable key, Iterator<Text> values, OutputCollector<IntWritable, Text> output, Reporter reporter) throws IOException {
          Integer numTrans = 0;
          Float totalSum = (float) 0;
          String strTotalSum;
          Integer minItem=11; //this variable store the minimual TransNumItems for each customer, and it is initially set as 11
          Integer transNumItem;

          while (values.hasNext()) {
                strTotalSum = values.next().toString();
                StringTokenizer tokenizer = new StringTokenizer(strTotalSum, ",");
              numTrans += Integer.parseInt(tokenizer.nextToken());
              totalSum += Float.parseFloat(tokenizer.nextToken());
              transNumItem = Integer.parseInt(tokenizer.nextToken());
              minItem = (transNumItem<minItem)?transNumItem:minItem;
          }

          Text transValue = new Text(numTrans.toString()+","+totalSum.toString()+","+minItem.toString());
          output.collect(key, transValue);
      }
    }

      public static class Map_2 extends MapReduceBase implements Mapper<LongWritable, Text, IntWritable, Text> {
      private IntWritable custID = new IntWritable();
      private Text value_final = new Text();
      String [] inMemArray = new String[20001];


      public void configure(JobConf conf) {
                Path pt=new Path("/user/david_zheng/project_1/Query_3/input_2/Customer.txt");

                String line = null;

                try {
                        FileSystem fs = FileSystem.get(new Configuration());
                        BufferedReader br=new BufferedReader(new InputStreamReader(fs.open(pt)));
                        while((line=br.readLine())!=null)
                        {
                                String record[] = line.split(",", -1);
                                //insert into array
                                inMemArray[Integer.parseInt(record[0])]=record[1]+","+record[4];

                        }
                } catch (Exception e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }

      }

      public void map(LongWritable key, Text value, OutputCollector<IntWritable, Text> output, Reporter reporter) throws IOException {
        String line = value.toString();

        StringTokenizer tokenizer = new StringTokenizer(line, "\t");
         Integer customerID;
          String custRecord;
          String transRecord;


          customerID=Integer.parseInt(tokenizer.nextToken());
          custRecord=inMemArray[customerID];
          transRecord=tokenizer.nextToken();

          //assign customerID to custID
          custID.set(customerID);
          //assign the value of the third attribute, TransTotal, to
          value_final.set(custRecord+","+transRecord);
          output.collect(custID, value_final);  //adds the key-value pair to the output
        }

      }


    public int run(String[] args) throws Exception {
        /* first job*/
      JobConf conf_1 = new JobConf(getConf(), Query_3.class);
      conf_1.setJobName("Query_3");
```

```
    conf_1.setOutputKeyClass(IntWritable.class);
    conf_1.setOutputValueClass(Text.class);

    conf_1.setMapperClass(Map_1.class);
    conf_1.setCombinerClass(Reduce_1.class);
    conf_1.setReducerClass(Reduce_1.class);

    conf_1.setInputFormat(TextInputFormat.class);
    conf_1.setOutputFormat(TextOutputFormat.class);

    FileInputFormat.setInputPaths(conf_1, new Path(args[0]));
    FileOutputFormat.setOutputPath(conf_1, new Path(args[1]));

    JobClient.runJob(conf_1);
      /* second job, map-only */
      JobConf conf_2 = new JobConf(getConf(), Query_3.class);
    conf_2.setJobName("Query_3");

    conf_2.setOutputKeyClass(IntWritable.class);
    conf_2.setOutputValueClass(Text.class);

    conf_2.setMapperClass(Map_2.class);

    conf_2.setInputFormat(TextInputFormat.class);
    conf_2.setOutputFormat(TextOutputFormat.class);

    FileInputFormat.setInputPaths(conf_2, new Path(args[1]));
    FileOutputFormat.setOutputPath(conf_2, new Path(args[2]));

    JobClient.runJob(conf_2);

    return 0;
  }


  public static void main(String[] args) throws Exception {
   int res = ToolRunner.run(new Configuration(), new Query_3(), args);
    System.exit(res);
  }
}
```