Problem Statement : Assume the datasets provided are very large (in millions) and you need to perform the same left join in a distributed environment, where the input data files are stored in Hadoop. Please provide a pseudo code using java map reduce to solve the left outer join problem mentioned above. You do not have to write the exact java map reduce code. But need to provide map reduce algorithm details using pseudo code.

Solution :

Java MapReduce solution using the classic map() and reduce() functions

Left Join :

To understand the concept of a left join, assume we have two types of data: “employee names” and “employee pay”. The employee_names data consists of (id, first_name, last_name,) and the employee_pay data includes employee identity information (id, salary, bonus), then:

employee names(id, first_name, last_name) say left table T1

employee_pay(id, salary, bonus) say right table T2

our goal is to find left join between above tables using java map reduce.

In SQL, we can express a left join as:

In SQL, the left join returns all the records from first table and matched records from second table. If there is no match from second table then only records from first table are returned.

SELECT field_1, field_2, ...
  FROM T1 **LEFT JOIN** T2

ON T1.K = T2.K;

Data Sets;

id,first_name,last_name. (**T1 table**)

```
59ea7840fc13ae1f6d000096,Benny,Ventom
59ea7840fc13ae1f6d000097,Cara,Motherwell
59ea7840fc13ae1f6d000098,Willem,Haresign
59ea7840fc13ae1f6d000099,Trish,Farlane
59ea7840fc13ae1f6d00009a,Camilla,Limpkin
59ea7840fc13ae1f6d00009b,Godwin,Caffrey
59ea7840fc13ae1f6d00009c,Elvera,Custed
59ea7840fc13ae1f6d00009d,Vinson,Farres
59ea7840fc13ae1f6d00009e,Kliment,Pitchford
59ea7840fc13ae1f6d00009f,Matty,Heater
59ea7840fc13ae1f6d0000a0,Juana,Begg
59ea7840fc13ae1f6d0000a1,Alix,Layus
59ea7840fc13ae1f6d0000a2,Tessa,Brandes
59ea7840fc13ae1f6d0000a3,Hedwig,Fishley
59ea7840fc13ae1f6d0000a4,Cordelia,Aubray
```

id,salary,bonus (**T2 table**)

```
59ea7840fc13ae1f6d000096,$84217.20,
59ea7840fc13ae1f6d000097,$54737.84,
59ea7840fc13ae1f6d000098,$92092.21,
59ea7840fc13ae1f6d000099,$86117.63,
59ea7840fc13ae1f6d00009a,$92821.29,
59ea7840fc13ae1f6d00009b,$60633.55,$1779.07
59ea7840fc13ae1f6d00009c,$94483.80,$4328.59
59ea7840fc13ae1f6d00009d,$93127.77,
59ea7840fc13ae1f6d00009e,$90581.06,
59ea7840fc13ae1f6d00009f,$86947.54,
59ea7840fc13ae1f6d0000a0,$80646.00,$2924.69
59ea7840fc13ae1f6d0000a1,$50423.64,$1495.33
59ea7840fc13ae1f6d0000a2,$60695.47,
```

59ea7840fc13ae1f6d0000a3,$85763.22,
59ea7840fc13ae1f6d0000a4,$60600.37,

id, first_name, last_name, salary, bonus (**Resulted table left join table**)
59ea7840fc13ae1f6d000096,Benny,Ventom,$84217.20,
59ea7840fc13ae1f6d000097,Cara,Motherwell,$54737.84,
59ea7840fc13ae1f6d000098,Willem,Haresign,$92092.21,
59ea7840fc13ae1f6d000099,Trish,Farlane,$86117.63,
59ea7840fc13ae1f6d00009a,Camilla,Limpkin,$92821.29,
59ea7840fc13ae1f6d00009b,Godwin,Caffrey,$60633.55,$1779.07
59ea7840fc13ae1f6d00009c,Elvera,Custed,$94483.80,$4328.59
59ea7840fc13ae1f6d00009d,Vinson,Farres,$93127.77,
59ea7840fc13ae1f6d00009e,Kliment,Pitchford,$90581.06,
59ea7840fc13ae1f6d00009f,Matty,Heater,$86947.54,
59ea7840fc13ae1f6d0000a0,Juana,Begg,$80646.00,$2924.69
59ea7840fc13ae1f6d0000a1,Alix,Layus,$50423.64,$1495.33
59ea7840fc13ae1f6d0000a2,Tessa,Brandes,$60695.47,
59ea7840fc13ae1f6d0000a3,Hedwig,Fishley,$85763.22,
59ea7840fc13ae1f6d0000a4,Cordelia,Aubray,$60600.37,

Implementation of Left Join in MapReduce:

MapReduce : find all employees (and their associated pay). We accomplish this using above SQL query.

To perform the left join operation with a MapReduce job, which will utilize two mappers (one for employee names and the other for employee pay) and whose reducer will emit a key-value pair. Using multiple mappers is enabled by the MultipleInputs class (note that if we had a single mapper, we would have
used Job.setMapperClass() instead):

```
import org.apache.hadoop.mapreduce.lib.input.MultipleInputs;
...
Job job = new Job(...);
...
Path employee_names = <hdfs-path-to-employee_names-data>;
Path employee_pay = <hdfs-path-to-employee_names-data>;

MultipleInputs.addInputPath(job,
                            employee_names,
                            TextInputFormat.class,
                            EmployeeNames.class);

MultipleInputs.addInputPath(job,
                            employee_pay,
                            TextInputFormat.class,

                            EmployeeJobs.class);
```

The core pieces of the left join data flow are as below:

Employee_names mapper

The Employee_names map() reads (id, first_name, last_name) and emits the key value pairs. we guarantee that Employee_names_id(s) arrive first. Added "L" to the value to identify employee names table.

```
/**
   * @param key is framework generated, ignored here
   * @param value is the id<,>first_name<,>last_name
   */

  map(key, value) {
          outvalue.set("L" + value.toString());
         context.write(outkey,outvalue)
 }
```

Employee_pay mapper

The Employee_pay map() reads (id, salary, bonus) and emits the key value pairs . We guarantee that employee_pay_id(s) arrive at the end. We added "R" to the value to identify Employee_pay.

```
 /**
  * @param key is framework generated, ignored here
  * @param value is the
  *    id<,>salary<,> bonus
  */
 map(key, value) {

         outvalue.set("B" + value.toString());
         context.write(outkey, outvalue);
}
```

The reducer phase :

Reducer copies all values for each group in memory ,keeping track of which record came from what data set

```
Public void reduce(Tetxt key, Iterable<Text> values, Context context) {

        // Clear.clear();
       listA.clear();
      listB.clear();

   // iterate through all our values, each record based on what it was
//tagged with. Make sure to remove tag.

While(values.hashNext()){
        Temp = values.next();
        If(temp.charAt(0) == 'L') {
                listA.add(new Text(temp.toString().subString(1)));
        } else if (temp.charAt(0) == 'R'){
                listB.add(new Text(Text(tmp.toString().subString(1)));
        }
}
  // Execute left join logic now that the lists are filled
 executeJoinLogic(context);
}
```

Left Join :
If the right list is not empty, join A with B.
If the right list is empty, output each record of A with an empty string.

executeJoinLogic :

```
for(Text  A : listB){
        //  if list B is not empty, join A and B
        If (!listB.isEmpty()) {
                for( Text B : listB){
                        context.write(A,B);
                }
        } else {
                // else output A by itself
                Context.write(A,Empty_TEXT);
        }
}
```