CS 361 – Homework Assignment 4
11:59pm, Friday April 29

This assignment will be graded on a 100 point scale:

- 50 points for correctly implementing the evaluate methods of the duplicate removal and join operations.
- 50 points for optimizing the projection, select, duplicate removal, and join operations
  - 10 points for some degree of optimization (beyond what I have done to start you off) on the projection operation
  - 10 points for some degree of optimization on the select operation
  - 10 points for some degree of optimization on the duplicate removal operation
  - 10 points for some degree of optimization on the join operation
  - 10 points for finishing at the median in the optimization competition (somewhere less than 10 for finishing below the median)
  - Up to 25 bonus points for finishing high in the optimization competition

Pick up the *assignment4-distro.zip* file from D2L. In it you will find the source code for the query interpreter discussed in the class notes of April 13:

The Tuple class – Tuple.java

The abstract base Table class – Table.java

The suite of classes that comprise data types for columns in Tables – ColumnValue.java, IntValue.java, FloatValue.java, StringValue.java,

The suite of classes for conditionals in Select operations – ComparisonConditional.java, Conditional.java, CondLeaf.java, ANDConditional.java, EQConditional.java, GTConditional.java, LTConditional.java

The five table operations that can appear in a query tree – DupsRemovedTable.java, JoinTable.java, ProjectionTable.java, SelectTable.java, TableInDB.java

A test harness – TestHarness.java

Compile everything with: `javac -cp opencsv-2.2.jar:. *.java`

Run the TestHarness with: `java -cp opencsv-2.2.jar:. TestHarness`

Note that initially only the first two tests will produce valid results. The other tests will crash until you correctly implement the operations. After you've implemented the join operation, be sure to un-comment the lines in the test harness that are presently commented out to prevent such a crash that will occur when a JoinTable is constructed as a child of another node/table in the query tree.

Six test tables and the OpenCSV library that I use to load the tables into ArrayLists – Table1, Table2, Table3, Table4, Table5, Table6 opencsv-2.2.jar

Javadocs for all the files In the docs directory of the distro, also at
`http://csf11.acs.uwosh.edu/cs361/query-opt-assignment/docs/index.html`

You will have to complete the implementation of four classes:

1. Complete the ProjectionTable class. The *evaluate* method is already done; the *optimize* method can be further enhanced using additional tree transformation heuristics.

```
/**
 * ProjectionTable - this class implements the projection operation of a
 * relational DB
 *
 */

public class ProjectionTable extends Table {

    Table tab_projecting_on;

    /**
     * @param t - the table from which attributes are projected
     * @param attribs_to_project - the names of the attributes that are to be projected
     *
     */
    public ProjectionTable(Table t, String attribs_to_project []) {

        super("Projecting from " + t.toString());
        tab_projecting_on = t;
        attr_names = attribs_to_project;
        attr_types = calculateProjectedAttributeTypes(attr_names);
    }

    public Table [] my_children () {
        return new Table [] { tab_projecting_on };
    }

    public Table optimize() {
        // One of the projection optimizations is done to illustrate the technique
        if (tab_projecting_on instanceof ProjectionTable) {
```

```
            // ? So how would you do the transformation that says: "Successive projections
            // can reduced to the final projection"
```

```
        }
        else {
```

```
            // ? And otherwise ?
```

```
        }
    }

    public ArrayList<Tuple> evaluate() {

        ArrayList<Tuple> tuples1 = tab_projecting_on.evaluate();
        ArrayList<Tuple> tuples_to_return = new ArrayList<Tuple>();
        ListIterator iterate_tuples = tuples1.listIterator(0);

        while (iterate_tuples.hasNext()) {
            Tuple tupleToProject = (Tuple) iterate_tuples.next();
            String[] projectedValues = new String[attr_names.length];
            for (int i=0; i < attr_names.length; i++) {
                projectedValues[i] = tupleToProject.get_val(attr_names[i]).toString();
            }
            Tuple projectedTuple = new Tuple(attr_names, attr_types, projectedValues);
            tuples_to_return.add(projectedTuple);
        }
        profile_intermediate_tables(tuples_to_return);
        return tuples_to_return;
    }

    /**
     * Determines the types of the specified attributes.
     * @param attributesToProject
     * @return String array of attribute types
     */
    String[] calculateProjectedAttributeTypes(String[] attributesToProject) {
        String[] attributeTypes = new String[attributesToProject.length];
        String[] attr_names = tab_projecting_on.attrib_names();
        String[] attr_types = tab_projecting_on.attrib_types();

        for (int i=0; i < attributesToProject.length; i++) {
                String projectedAttribute = attributesToProject[i];
                String projectedAttributeType = null;
                for (int j = 0; j < attr_names.length; j++) {
                        if (projectedAttribute.equals(attr_names[j])) {
                                projectedAttributeType = attr_types[j];
                                break;
                        }
                }
                attributeTypes[i] = projectedAttributeType;
        }
        return attributeTypes;
```

```
    }
}
```

2. Complete the implementation of the SelectTable class. The *evaluate* method is complete, but the *optimize* method is merely an empty shell.

```
/**
 * SelectTable - this class implements the select operation of a
 * relational DB
 *
 */

public class SelectTable extends Table {

    Table tab_selecting_on;
    Conditional select_cond;;

    /**
     * @param t - the table we are selecting from
     * @param c - the conditional used to make the selection
     *
     */
    public SelectTable(Table t, Conditional c) {

        super("Select on " + t.toString() + " on condition " + c.toString());
        tab_selecting_on = t;
        select_cond = c;
        attr_names = t.attrib_names();
        attr_types = t.attrib_types();
    }

    public Table [] my_children () {
        return new Table [] { tab_selecting_on };
    }

    public Table optimize() {
        return this;
    }

    public ArrayList<Tuple> evaluate() {

        if (select_cond instanceof ANDConditional) {
            for (int i = 0; i < ((ANDConditional) select_cond).my_conds.length; i++)
                ((ANDConditional)select_cond).my_conds[i].set_both_leaves_table(tab_selecting_on);
        }
        else
            ((ComparisonConditional) select_cond).set_both_leaves_table(tab_selecting_on);
        ArrayList<Tuple> tuples1 = tab_selecting_on.evaluate();
        ArrayList<Tuple> tuples_to_return = new ArrayList<Tuple>();
        ListIterator iterate_tuples = tuples1.listIterator(0);

        while (iterate_tuples.hasNext()) {
            Tuple x = (Tuple) iterate_tuples.next();
            if (select_cond.truthVal(x)) {
                tuples_to_return.add(x);
            }
        }
        profile_intermediate_tables(tuples_to_return);
        return tuples_to_return;
    }
}
```

3. Complete the implementation of the DupsRemovedTable – both *evaluate* and *optimize* are empty shells. In *evaluate*, use either the sort-based or hash-based algorithm discussed in class. Identify which one you've used in an opening comment.

```
/**
 * DupsRemovedTable - this class implements the duplicate removal operation of a
 * relational DB
 *
 */

public class DupsRemovedTable extends Table {

    Table tab_dups_removed_from;

    /**
     * @param t - the table from which duplicates are to be removed
     *
     */
    public DupsRemovedTable(Table t) {

        super("Removing duplicates from " + t.toString());
        tab_dups_removed_from = t;

    }

    public Table [] my_children () {
        return new Table [] { tab_dups_removed_from };
    }

    public Table optimize() {
        // Right now no optimization is done -- you'll need to improve this
        return this;
    }

    public ArrayList<Tuple> evaluate() {
        ArrayList<Tuple> tuples_to_return = new ArrayList<Tuple>();

        // Here you need to add the correct tuples to tuples_to_return
        // for this operation

        // It should be done with an efficient algorithm based on
        // sorting or hashing

        profile_intermediate_tables(tuples_to_return);
        return tuples_to_return;

    }

}
```

4. Complete the implementation of the JoinTable – both *evaluate* and *optimize* are empty shells. In *evaluate*, use either the sort-based or hash-based algorithm discussed in class. Identify which one you've used in an opening comment.

```
/**
 * JoinTable - this class implements the join operation of a
 * relational DB
 *
 */

public class JoinTable extends Table {

    Table first_join_tab;
    Table second_join_tab;

    /**
     * @param t1 - One of the tables for the join
     * @param t2 - The other table for the join.  You are guaranteed
     * that the tables do not share any common attribute names.
     * @param c - the conditional used to make the join
     *
     */
    public JoinTable(Table t1, Table t2, Conditional c) {

        super("Joining " + t1.toString() + " " + t2.toString() + " on condition " + c.toString());
        first_join_tab = t1;
        second_join_tab = t2;
    }

    public Table [] my_children () {
        return new Table [] { first_join_tab, second_join_tab };
    }

    public Table optimize() {
        // Right now no optimization is done -- you'll need to improve this
        return this;
    }

    public ArrayList<Tuple> evaluate() {
        ArrayList<Tuple> tuples_to_return = new ArrayList<Tuple>();

        // Here you need to add the correct tuples to tuples_to_return
        // for this operation

        // It should be done with an efficient algorithm based on
        // sorting or hashing

        profile_intermediate_tables(tuples_to_return);
        return tuples_to_return;
    }
}
```

How to finish and submit your work to the dropbox:

- To participate in the optimization contest be sure you have not forgotten to insert the appropriate call to the *Table* class *profile_intermediate_tables* method immediately before you return your ArrayList of Tuples from all of the *evaluate* methods.
- Prepare a README text file in which you specify any test cases for which your implementation crashes. I will be sure not to test those test cases for your code. If I run a test case that crashes and you haven't forewarned me of that, a 10 point penalty will be imposed for each such test case.
- Archive the:
  - The four Java source files you have implemented
  - README file

  in a zip file called *assign4.zip* and deposit that zip file in the dropbox by the deadline.
  Just include these four Java source code files and your README file in this zip volume. If you instead add a bunch of files that clutter up the archive with files that are not supposed to be submitted, 10 points will be deducted. In particular, after I unzip your file, I will *cd* to the directory that contains the files, copy the other files you originally received to that directory, and then issue the command:

  ```
  javac -cp opencsv-2.2.jar:. *.java
  ```

  If any of your four Java source code files fail to compile correctly, the game is over and you will receive a *very poor* grade.
  I will then do the test runs by issuing:

  ```
  javac -cp opencsv-2.2.jar:. TestHarness
  ```

  I will assume that you haven't put your code in a package. If you use an IDE like Eclipse or Netbeans, they might do this "automatically" for you. So before you submit, remove any package statements added by your IDE and test compiling and running as illustrated above. If your code has package statement in it, it will not compile or run correctly.