CSE 562: Project 1 (3 pages)

Due Date: No later than 12:01 AM on Monday, February 18
Submission Instructions: Submit your assignment as a compressed (tar.gz, or zip) copy of the `sql` project directory using the departmental submit scripts. Upload your file to `timberlake.cse.buffalo.edu`, then log in and run the command:

```
/util/bin/submit_cse562 [filename]
```

Testing: I will delete the build directory (if it exists), run `‘make‘` at the root level, and then test each component of the project as described below using the `build` directory as the *only* element of the classpath. If you prefer using ant, you can replace the makefile with one that invokes ant.

Interpreting Relational Algebra (60 pts)

The class `edu.buffalo.cse.sql.plan.PlanNode` in the project directory corresponds to a single relational operator. This class forms the basis for a tree of relational operators that together express a query. Most of the remaining classes in the same package correspond to instantiations of specific relational operators (Aggregate, Join, Project, Select, Union, and File Scan).

In this part of the assignment, you will be handed a PlanNode that corresponds to the root of a query (most of which reference data files located in the test directory). Your goal is to evaluate this query and provide a response.

The test classes in package `edu.buffalo.cse.sql.test` will help you. Each contains a main function that invokes a static method on `edu.buffalo.cse.sql.Sql`, which you should define (a stub is provided in the project code):

```
public static List<Datum[]> execQuery(
    Map<String, Schema.TableFromFile> tables,
    PlanNode q
  )
```

`tables` defines a set of mappings from Table Name (as referenced by the File Scan operators) to metadata about the table (see `edu.buffalo.cse.sql.Schema` for details). This includes a path to the file containing the data (relative to the root of the sql directory) for the relation.
`q` contains the query.
You should return a `List` of rows, each encoded as an array of `edu.buffalo.cse.sql.data.Datum`s.

Run the command

```
java -cp build edu.buffalo.cse.sql.test.[TESTNAME]
```

If your evaluator produces the correct results, this should produce the output:

`Passed RA test [TESTNAME][-test#]`

Grading: Successfully passing all the tests provided in the tests package earns you a 55 on this portion of the assignment. The remaining 5 points can be obtained by successfully evaluating several additional test cases that we will run on your code. Note that the provided test cases intentionally do not cover several possible combinations and uses of the relational algebra operators.

Translating SQL to Relational Algebra (40 pts)

For this portion of the assignment, you will be given a `java.io.File` pointing to one of the SQL files in the test directory. You will need to parse in the SQL expression in the file and generate a corresponding relational algebra expression, which you will then be expected to evaluate.

You are responsible for correctly parsing queries of the form:

```
SELECT target_list
FROM relation_list
WHERE condition
GROUP BY group_by
```

The `FROM`, `WHERE`, and `GROUP BY` clauses are optional and may be omitted from some queries.

`target_list` is a list of comma-separated expressions, each in one of the following forms:

```
expr [[AS] colname]
agg_fn(expr [[AS] colname])
COUNT(*)
```

Here `expr` is an arithmetic expression as defined below, and `colname` is an optional column name. If `colname` is omitted, and expr is a single variable, you should assume the variable's name. Otherwise, use an arbitrary name.

`agg_fn` is one of `SUM`, `AVG`, `MIN`, or `MAX`

Expressions can be

- Strings (surrounded by single quote marks ('), using the backslash (“) as an escape character for single-quote, and the double-backslash as an escape character for backslash itself).
- Boolean expressions using "AND" "OR" "NOT" as operators, "TRUE" and "FALSE" as literals, and supporting all six basic comparison operations = <, <=, >=, >, <>. You *do not* need to implement any of the string or nested query predicates.
- Arithmetic expressions using "+", "-", "*", "/", and using floats and integers as literals.

Note the `edu.buffalo.cse.sql.data.Datum` class, which can help you to represent literals.
`relation_list` is a comma-separated list of relation names. You *do not* need to implement "JOIN", "NATURAL JOIN", or nested queries for this assignment.
`condition` is an expression that evaluates to a boolean value
`group_by` is a list of column names.

You will also be responsible for parsing in table constructors. These have the form

```
CREATE TABLE rel_name (rel_spec) FROM FILE 'file_path' USING CSV;
```

Where `rel_name` is the relation name, `file_path` is a path (relative to the `sql` directory) to the data file. All data files for this assignment will be in CSV format.
`rel_spec` is a comma-separated list of field identifiers, each specified as a name, followed by one of `int`, `float`, or `string`.

The test classes in package `edu.buffalo.cse.sql.test` will once again help you to test. If the main function of one of these classes is invoked with the single argument `sql`, it will call a different static method on your `Sql` class.

```
public static List<List<Datum[]>> execFile(
    File program
  )
```

`program` is a file referencing one of the SQL files in the `test` directory. Parse in this file, execute the queries that appear inside, and return the query results *in the order that they appear in the file.*

Run the command

```
java -cp build edu.buffalo.cse.sql.test.[TESTNAME] sql
```

If you parse and evaluate the query correctly you will receive the output

```
Passed SQL test [TESTNAME][-test#]
```

Grading: Successfully passing all the sql tests provided in the tests package earns you a 35 on this portion of the assignment. The remaining 5 points can be obtained by successfully evaluating several additional test cases that we will run on your code.