Database System Implementation CSCI 421
Group Project Phase 3

v1.0

1 Phase Description

This is the third phase of the semester long group project.

In this phase you will be implementing the DML parser. The tool to query data in the database.

There are a few basic rules when implementing this phase:

- You must create and use an instance of the storage manager you created in phase 1. You will not use all of its features in this phase.
- You can assume the database location path exists and is accessible.
- You must use the catalog structure implemented in the previous phase.

2 Phase Layout

You will be finishing the DML statements of the Query Processor.

3 Query Processor

In this phase you will finish implementing the `select` command.

4 `select` Command

In this phase you will implement the `select` query.

- Each query can be one line or multiple lines.
- The newline character is to be considered the same as a space.
- Multiple spaces are to be considered a single space; but where spaces are shown below at least one space will exist there.
- All statements will end with a semi-colon.

4.1 `select` queries

These queries will look very similar to SQL, but the format is going to be changed to help reduce parsing complexity.

This will display the results to the screen to the user in an easy to read format.

When accessing multiple tables in a query, common column names in the tables must be qualified with the table name.

Example: Both tables `foo` and `bar` have an attribute called `id`.

This is not allowed:

```
select id
from foo, bar
where id > 5 and id = id;
```

As it is ambiguous on which `id` is being used.

Which `id` is being referenced. To clarify this:

```
select foo.id
from foo, bar
where foo.id > 5 and foo.id = bar.id;
```

The typical format of a query is:

```
select <a_1>, ..., <a_N>
from <t_1>, ..., <t_N>
where <condition(s)>
orderby <a_1>
```

Lets look at each part:

- `select`: All DML queries that start with this will be considered to be trying to query data the database. Keyword.
- `select <a_1>, ..., <a_N>`: This selects the columns to output in the final result. If two tables have the same column name the column must be referred to by table name and column name in the query. If this contains a `*` only it will output all of the columns; the `*` and attributes together is an error. There must be at least one column or the `*` listed.
- `from <t_1>, ..., <t_N>`: this will preform a Cartesian product of the tables whose names are listed. There can be between 1 and N tables. Note: it might be advantageous to do the Cartesian product after evaluating the where clauses. `from` is a keyword.
- `where <condition>`: A condition where a tuple should be in the output. If this evaluates to true the tuple is placed in the output table; otherwise it is not. See prior phase for information about conditionals. If there is no `where` clause it is considered to be a `where true` and all tuples get outputted.
- `orderby <t_1>`: this will order the output table by the column listed in the order they are listed (like in SQL). Attributes in the order by must also appear in the select clause. `orderby` is considered a keyword. Not all queries have to have an order by.

4.2 Example Queries

```
select * from foo;

select name, gpa
from student;

select name, dept_name
from student, department
where student.dept_id = department.dept_id;
```

```
    select *
    from foo
    orderby x;

    select t1.a, t2.b, t2.c, t3.d
    from t1, t2, t3
    where t1.a = t2.b and t2.c = t3.d
    orderby t1.a;
```

5 Conditionals

This section will outline the process of evaluating conditionals in the `where` clause.

Conditionals can be a single relational operation or a list of relational operators separated by `and` / `or` operators. `and` / `or` follow standard computer science definitions:

- <a> `and` <b>: only true if both `a` and `b` are true.
- <a> `or` <b>: only true if either `a` or `b` are true.

`and` has a higher precedence than `or`. Items of the same precedence will be evaluated from left to right.

Example:

```
    x > 0 and y < 3 or b = 5 and c = 2
```

You would first evaluate `x > 0 and y < 3`, then `b = 5 and c = 2`. Then `or` their results.

This project will only support a subset of the relational operators of SQL:

- = : if the two values are equal.
- > : greater than.
- < : less than.
- >= : greater than or equal to.
- <= : less than or equal to.
- != : if the two values are not equal.

Relational operators will return true / false values. You can use the Java built in comparison functionality to compare values. The left side of a relational operator must be an attribute name; it will be replaced with its actual value at evaluation. The right side must be an attribute name or a constant value; no mathematics. The data types must be the same on both sides on the comparison.

Examples:

```
    foo > 123
    foo < "foo"
    foo > 123 and baz < "foo"
    foo > 123 or baz < "foo bar"
    foo > 123 or baz < "foo" and bar = 2.1
    foo = true
    foo = baz
```

Strings, unless quoted or true/false, are to be considered column names.

6 Project Constraints

This section outlines details about any project constraints or limitations.

Constraints/Limitations:

- Everything, except for the values in Strings (char and varchar), is case-insensitive; like SQL. Anything in double quotes is to be considered a String. String literals will be quoted and can contain spaces. Example: `"foo bar"` is a string literal. `foo` is a name; not a string literal. The quotes do not get stored in the database. They must be added when printing the data.
- You must use a Java and the requirements provided.
- Your project must run and compile on the CS Linux machines.
- Submit only your source files in the required file structure. Do not submit and IDE directories or projects.
- Your code must run with any provided tests cases/code. Failure to do so will result heavy penalties.
- Any errors, such as file reading/writing errors, should be handled with a useful error message printed to system error stream. The user can determine how to handle the error based on the return of the functions.
- Words such as `Integer`, `create`, `table`, `drop`, etc are considered key words and cannot be used in any attribute or table name.
- Data must be checked for validity. Examples:
  - type matching
  - not nulls
  - primary keys
- Any changes to data in the database must keep the database consistent. This means that all not null, foreign keys, and primary keys must be observed.

7 Grading

Your implementation will be grading according to the following:

- (100%) Select statements
  - (15%) `select` functionality
  - (30%) `from` functionality
  - (30%) `where` functionality
  - (25%) `orderby` functionality

Penalties:

- (-50% of points earned) Data is written as text not binary data.

- (-25% of points earned) Does not use storage manager to handle hardware access. Catalog reading/writing to hardware does not need to be handled by the storage manager.
- (up to -50% of points earned) Does not work with any provided testing files.
- (up to -100% of points earned) Does not compile on CS machines.

Penalties will be based on severity and fix-ability. The longer it takes the grader to fix the issues the higher the penalty.

Examples:

- Code does not compile due to missing semicolon: -5%
- Code does not compile/run with testers due to missing functions or un-stubbed functions: -10%
- Multiple syntax errors causing issues compiling issues and takes over 30 mins to fix: -100%
- Multiple Crashes with provided testers that take an extended time to fix: -50%

These are just examples.The basic idea is ”Longer to fix, more points lost. Eventually give up, assign a zero.”

8 Submission

Zip any code that your group wrote in a file called `phase3.zip`. Maintain any required package structure. Do NOT submit any provided code. The grader will use their own versions.

Submit the zip file to the Phase 1 Assignment box on myCourses. No emailed submissions will be accepted. If it does not make it in the proper box it will not be graded.

The last submission will be graded.

There will be a 48 hour late window. During this late window no questions will be answered by the instructor. No submissions will be accepted after this late window.

9 Tips and Tricks

Here are a few tips and tricks to help you:

- START EARLY. Some of this can be tricky and you will have questions.
- Come to the in class project sessions. Instructor will be there to help.
- Design, Design, Design. ”Every hour in design can save 8 hours of coding.” is a common saying.
- If you follow the interfaces and use them as intended, the work can be divided up.
- Design smaller helper functions to make life easier.
- Read the entire document and the provided code commenting.
- Looking ahead at future phases might be helpful.
- Break things up! Make functions to parse each part

  - `parseSelect`
  - `parseWhere`
  - `parseFrom`
  - etc
- Trees might help!