Course Project

An Interpreter for SSQL

Part I (20 points)

Introduction to SSQL

- SSQL(short for Simplified SQL) is a subset of features in standard SQL language.
- Four operations supported in SSQL:
  - Create a new table
  - Insert a row into an existed table
  - Delete rows from an existed table
  - Query on an existed table
- No join, aggregation, complex constraints ……

Examples

- Standard SQL (MySQL)

```
SELECT S.sid AS id, SUM(C.score) AS score
FROM Student S, Courses C
WHERE S.sid == C.sid && S.age > 18
GROUP BY id
ORDER BY score
LIMIT 100;
```

- SSQL

```
SELECT sid, age
FROM Student
WHERE age > 15 && age < 18;
```

Context Free Grammar

ssql_stmt → *create_stmt*
| *insert_stmt*
| *delete_stmt*
| *query_stmt*

Context Free Grammar

- Create Statement

create_stmt → **create table id (***decl_list***);**
decl_list → *decl* | *decl_list***,** *decl*
decl → **id int** *default_spec* | **primary key (***column_list***)**
default_spec → **default = num** | ε
column_list →**id** | *column_list***, id**

- **id** (identifier) is a sequence of digits, underline and letters. All identifiers should start with a letter or an underline. The maximum length of an identifier is 64.
- **num** (number) is a sequence of digits. (of 32-bits)
- Reserved keywords are case-insensitive. (See Appendix)
- If the default value is not specified, 0 is used implicitly.

Context Free Grammar

- Create Statement

create_stmt → **create table id (***decl_list***);**
decl_list → *decl* | *decl_list***,** *decl*
decl → **id int** *default_spec* | **primary key (***column_list***)**
default_spec → **default = num** | ε
column_list →**id** | *column_list***, id**

```
CREATE TABLE Student(sid INT,
                     age INT DEFAULT = 18,
                     PRIMARY KEY (sid));
```

Context Free Grammar

- A *valid* create statement is:
  - can be derived from the context free grammar
  - no duplicate column names

```
CREATE TABLE Student(sid INT,
                     age INT DEFAULT = 18,
                     PRIMARY KEY (sid),
                     age INT);
// strange if we have two or more columns with same name
```

Context Free Grammar

- A *valid* create statement is:
  - can be derived from the context free grammar
  - no duplicate column names
  - no two or more primary key declarations

```
CREATE TABLE Student(sid INT,
                     PRIMARY KEY (sid),
                     age INT DEFAULT = 18,
                     PRIMARY KEY (age));
// it is possible to have two primary key declarations
// which set different columns as primary key, which should be unique
```

Context Free Grammar

- A *valid* create statement is:
  - can be derived from the context free grammar
  - no duplicate column names
  - no two or more primary key declarations
  - primary key contains only columns in the table

```
CREATE TABLE Student(sid INT,
                     PRIMARY KEY (sid, height),
                     age INT DEFAULT = 18);
// height is not a column of Student(sid, age)
```

Context Free Grammar

- A *valid* create statement is:
  - can be derived from the context free grammar
  - no duplicate column names
  - no two or more primary key declarations
  - primary key contains only columns in the table
  - table name doesn't match with any existed table

Context Free Grammar

- A *valid* create statement is:
  - can be derived from the context free grammar
  - no duplicate column names
  - no two or more primary key declarations
  - primary key contains only columns in the table
  - table name doesn't match with any existed table
  - # of columns in a table should <= 100

Context Free Grammar

- A *valid* create statement is:
  - can be derived from the context free grammar
  - no duplicate column names
  - no two or more primary key declarations
  - primary key contains only columns in the table
  - table name doesn't match with any existed table
  - # of columns in a table should <= 100
  - # of columns in primary key declaration <= 100

Context Free Grammar

- A *valid* create statement is:
  - can be derived from the context free grammar
  - no duplicate column names
  - no two or more primary key declarations
  - primary key contains only columns in the table
  - table name doesn't match with any existed table
  - # of columns in a table should <= 100
  - # of columns in primary key declaration <= 100
- If a create statement is successfully executed, materialize a table with the specified schema.

Context Free Grammar

- Insert Statement

insert_stmt → **insert into id(***column_list***) values (***value_list***);**
value_list → **num** | *value_list*, **num**

```
INSERT INTO Student(sid, age) VALUES(1111, 18);
```

Context Free Grammar

- A *valid* insert statement is:
  - can be derived from the context free grammar
  - the table should exist
  - no duplicate columns (see the example below)

```
INSERT INTO Student(sid, age, age) VALUES(1111, 18, 19);
// it is possible to set different values for the same column
```

Context Free Grammar

- A *valid* insert statement is:
  - can be derived from the context free grammar
  - the table should exist
  - no duplicate columns (see the example below)
  - all columns should be in the schema of the table

```
INSERT INTO Student(sid, height) VALUES(1111, 18);
// height is not a column in the schema of Student(sid, age)
```

Context Free Grammar

- A *valid* insert statement is:
  - can be derived from the context free grammar
  - the table should exist
  - no duplicate columns (see the example below)
  - all columns should be in the schema of the table
  - # of columns should equal to # of values

```
INSERT INTO Student(sid, age) VALUES(1111, 18, 19);
// 1111 → sid,  18 → age,  19 → ??
```

Context Free Grammar

- A *valid* insert statement is:
  - can be derived from the context free grammar
  - the table should exist
  - no duplicate columns (see the example below)
  - all columns should be in the schema of the table
  - # of columns should equal to # of values
  - no primary key constraint violation

Context Free Grammar

- A *valid* insert statement is:
  - can be derived from the context free grammar
  - the table should exist
  - no duplicate columns (see the example below)
  - all columns should be in the schema of the table
  - # of columns should equal to # of values
  - no primary key constraint violation
- For a column without specified value, default value is used. If an insert statement is executed successfully, a new row will be inserted into the table.

Context Free Grammar

- Delete Statement

delete_stmt → **delete from id** *where_clause*;
where_clause → **where** *conjunct_list* | *ε*
conjunct_list → *bool* | *conjunct_list* **&&** *bool*
bool →*operand rop operand*
operand → **num** | **id**
rop → **<>** | **==** | **>** | **<** | **>=** | **<=**

```
DELETE FROM Student WHERE age < 18 && age > 14;
DELETE FROM Student;     // delete all
```

Context Free Grammar

- A *valid* delete statement is:
  - can be derived from the context free grammar
  - the table should exist
  - columns occurring in the where clause (if any) should be in the schema of the table

```
DELETE FROM Student WHERE age < 18 && height > 180;
// height is not a column in the schema of Student(sid, age)
```

Context Free Grammar

- A *valid* delete statement is:
  - can be derived from the context free grammar
  - the table should exist
  - columns occurring in the where clause (if any) should be in the schema of the table

- If there is a where clause, delete all rows whose where clause is evaluated to be TRUE. Otherwise, delete all rows in the table.

Context Free Grammar

- Query Statement

query_stmt → **select** *select_list* **from id** *where_clause***;**
select_list → *column_list* | ***

```
SELECT sid, age FROM Student WHERE age < 18;
SELECT sid, age FROM Student;   // return all rows
SELECT * FROM Student;      // select all columns and return all rows
```

Context Free Grammar

- A *valid* query statement is:
  - can be derived from the context free grammar
  - the table should exist
  - all columns (except *) in the select list should be in the schema of the table

```
SELECT sid, height FROM Student WHERE age < 18;
// there isn't a column called “height” in Student
```

Context Free Grammar

- A *valid* query statement is:
  - can be derived from the context free grammar
  - the table should exist
  - all columns (except *) in the select list should be in the schema of the table
  - columns occurring in the where clause (if any) should be in the schema of the table

```
SELECT sid, age FROM Student WHERE height < 180;
// there isn't a column called “height” in Student
```

Context Free Grammar

- If a where clause is present, those rows whose where clauses are evaluated to FALSE should be omitted. Otherwise, none should be omitted.
- If '*' is present in the select list, all columns should be returned. Otherwise, return only those columns specified in the select list.

Implementation Requirements

- C/C++ only, no external tools (flex, bison, and the like) except STL
- Lexer – implement using a deterministic finite automaton
- Parser – implement a predictive parser
- Clearly structured and well readability
- Plagiarism prohibited
- 3 – 5 students a group

Program Input

- Your program has to implement in such a way that we can pass *a file system path to a ASCII file containing several SSQL statements* to your program when we run your program. (refer to this)
  - e.g.

```
yikai@yikai-ThinkPad-T400:~$ ./ssql "/home/yikai/Code/test.txt"
```

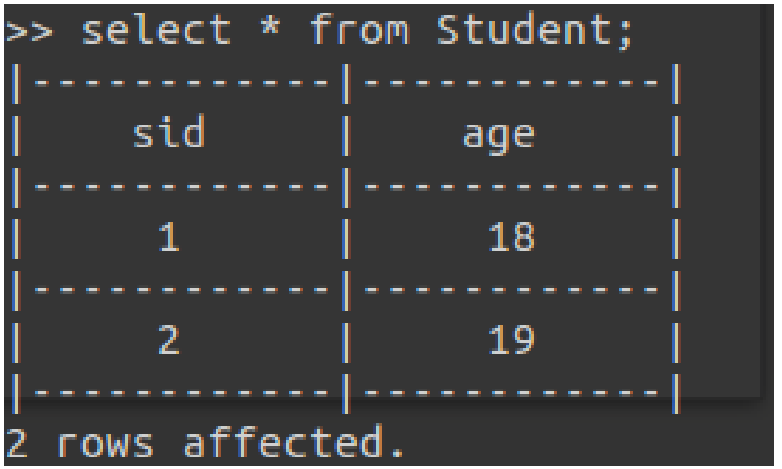
- Your program has to read the file and interpret the statements in the file.

Errorneous Input

- The input may contain ill-formed statements. Your program should be able to print meaningful error prompts when it encounters one.
- When an error is encountered, stop parsing or executing the current statement and continue to parse and execute the next statement.

Program Output

- When an error is encountered, print meaningful error prompts. (which line, which column, what's the error …)
- If the execution succeeds, print a message telling the success.
- Specially, for a query, print the result in a neat way. The effect should be similar to:



Test

- You should design test samples to test your program carefully and write a test document.
- A test sample contains:
  - a single statement or several statements to test the program
  - what is this test sample for
  - what's the expected output
  - does the program work correctly
- TAs will have their own test samples to test your program.

Things to Submit

Deadline: Jan, 3, 2015!

- Source codes
- Design document: how you implement?
  - Time and space complexity
  - What your group members did?
  - Group member infos (name, student id)
  - ……
- Test document: does the interpreter work?
  - Test samples and screenshots
- Makefile: build your program
- Total as a tarball, i.e. compiler_{groupid}.[zip, rar, tar.gz]

Part II (10 points)

Expression

- The interpreter is required to support the following operations:
  - Arithmetical operators: +, -, *, /, unary -, unary +
  - Relational operators: <, >, <>, ==, >=, <=
  - Logical operators: &&, ||, !.
- See the context free grammar in the following slides.

Expression

default_spec and *value_list* overridden:

default_spec → **default =** *simple_expr* | ε
value_list → *simple_expr* | *value_list*, *simple_expr*
simple_expr → *simple_term*
| *simple_expr* **+** *simple_term*
| *simple_expr* **-** *simple_term*
simple_term → *simple_unary*
| *simple_term* ***** *simple_unary*
| *simple_term* **/** *simple_unary*
simple_unary → **(***simple_expr***)** | **-***simple_unary* | **+***simple_unary* | **num**

```
CREATE TABLE Student(sid INT DEFAULT = 6 * 3 / 4 + 1,
                                age INT DEFAULT = 32 * 3 – 6 / 2);
INSERT INTO Student(sid, age) VALUES(6 * 3 / 4 + 1, 32 * 3 – 6 / 2);
```

Expression

where_clause and *bool* overridden:

where_clause → **where** *disjunct* | ε
disjunct → *conjunct* | *disjunct* **||** *conjunct*
conjunct → *bool* | *conjunct* **&&** *bool*
bool → **(***disjunct***)** | **!***bool* | *comp*
comp → *expr rop expr*
expr → *term* | *expr* **+** *term* | *expr* **-** *term*
term → *unary* | *term* ***** *unary* | *term* **/** *unary*
unary → **(***expr***)** | **-***unary* | **+***unary* | **id** | **num**

```
DELET FROM Student
WHERE age + 7 > 19 + 6 && sid <> 6 / 3 – 2;

SELECT * FROM Student
WHERE age + 7 > 19 + 6 && sid <> 6 / 3 – 2;
```

Expression

- This problem should also be described in the design document and test document.

- Note that it is possible to have “division by 0” error when the program executes. You should take care of this case carefully such that the program won't crash.

Q&A

Feel free to contact me if

- you have any problem or
- you find any bug in this project

echo_evenop@yahoo.com

Appendix

- Reserved keywords are:

  create, table, int, default, primary, key, insert, into, values, delete, from, where, select