StormSQL

SQL server in C++

This project has been created only for educational purposes. It's not meant to be used in any sort of production environment.

However, I think it's a great way to learn the dark ways of C++, not only by coding it, but also by inspecting the code and trying to understand it.

You can use, distribute and do whatever you want with this code. The only requirement is that you ***do not*** try to present it as your own.

Project information

This is a Visual Studio 2012 solution and has been tested only on Windows platform. However, **no** OS-specific code has been used so it should be possible to compile it on other OS with (for example) GCC.

The code is completely created using OOP paradigms. No external dependencies (libraries) should be needed. Only standard STL classes are used.

The solution contains **3** projects:

- **SQL** is a statically linked library project. It contains the core of the SQL server
- **SQLConsole** is a console interface to the **SQL** library. It loads/stores a single database in a file, reads queries from the console, executes them against the database and shows the results. **When entering queries in the console you NEED TO end each query with ';'** I made it this way so you can span a query multiple lines (as in the mysql console).
- **UnitTests** contains ~40 tests *(at the time of writing)* which should cover ~80% of the code according to VS's analyzer.

Features and supported queries

Because no one likes Backus-Naur forms I will just write a few examples which I think are most descriptive and explain the functionality.

Some notes

- The SQL keywords are case-insensitive *(SELECT == select == sElECT)*.
- The identifiers can be enclosed with back ticks but it's not needed *(`students` == students)*.
- The identifiers (table names, column names etc.) and the data in char(*) columns are case sensitive.
- In string literals the single and double quotes are interchangeable as long as the opening and closing quotes are the same. The backward slash escapes a quote of the same type. *('test' == "test")*

Create

For the next examples let's assume that we've created a table with:

```
CREATE TABLE students (fnum int, firstName char(25), lastName char(25), age byte,
semester byte);
```

There are no limits on the maximum size of the char columns other than that it needs to fit in a 32bit integer.

The supported column types are:

- **int** - 32bit signed integer
- **uint** - 32bit **unsigned** integer
- **byte** - One byte (-127 to 128) integer
- **char(size)** - String with maximum of `size` characters. **Only ASCII is supported**

Show Tables

Lists all tables in the database along with the row count for each one.

```
SHOW TABLES;
```

Show Create

Shows the create table statement that will create the same table.

```
SHOW CREATE TABLE students;
```

Drop Table

Deletes a table

```
DROP TABLE students;
```

Select

```
SELECT * FROM students;
SELECT * FROM students WHERE firstName = 'Georgy' AND lastName = 'Angelov';

-- Expressions in where clause
SELECT * FROM students WHERE strcat(strcat(firstName, " "), lastName) = 'Georgy
Angelov';

-- Expressions in result table (computed columns) in addition to the actual columns
SELECT *, strlen(firstName) + strlen(lastName) + 1 AS nameLength FROM students
WHERE ...;

-- Chained expressions
SELECT substr(firstName, 0, 1) AS firstLetterF, substr(lastName, 0, 1) as firstLetterL,
strcat(firstLetterF, firstLetterL) AS initials FROM students;

-- Comparison operators (<, <=, >, >=, =, !=) work on integers and strings
SELECT * FROM students WHERE lastName > 'Angelov';
```

There are two types of expressions - int and str. The boolean type is simulated with an int (0 or 1) All function names are also case-insensitive. Conversion functions are available:

- **toStr()** returns the string representation of an int (1234 -> "1234")
- **toInt()** returns the integer representation of a string ("1234" -> 1234)
- **toBool()** returns 0 (if is 0 or is "") or 1 (if != 0 or strlen() > 0)

Example:

```
-- Append the name length after the actual first name
-- Without the toStr an error will be shown
-- because the return type of strlen is int
SELECT strcat(firstName, toStr(strlen(firstName)) AS nameWithLen FROM
students ...;
```

JOINs are supported!

```
SELECT * FROM students JOIN courses ON fnum = sfnum;
SELECT * FROM students JOIN courses ON fnum < sfnum + 2;
```

Insert

Insert supports only the following syntax:

```
INSERT INTO students VALUES (1234, 'Georgy', 'Angelov', 20, 2);
```

The column names should not be specified (as they can't have default values and all will be listed anyway).

Update

The WHERE clause for the UPDATE statement is **required**. This is my oppinion of how this should be in SQL. It's easy to forget WHERE when you write a complex query and you will break your entire table with a mistake that can be easily avoided if WHERE is required.

```
UPDATE students SET fnum = 4321, firstName = 'George' WHERE lastName =
'Angelov';

-- The values after '<columnName> = ' are expressions (as in SELECT and the
WHERE clause)
UPDATE students SET fnum = fnum + 1, firstName = strcat('-test-', firstName)
WHERE ...;

-- WHERE is required, however if we want to change ALL rows:
UPDATE students SET age = age + 1 WHERE true;
-- or 1 = 1, whatever evaluates to boolean true
```

Delete

As with UPDATE - the WHERE clause here is also required.

```
DELETE FROM students WHERE fnum = 1234;

-- Delete all rows
DELETE FROM students WHERE true;
```

Want more information?

Then email me - georgyangelov@gmail.com