

CPP Chapter.12 SQLite Tutorial

Contents

CPP Chapter.12 SQLite Tutorial	1
SQLite and C++	1
Setup the Project: SQLite with C++	2
Tutorial 1: Connect to an SQLite Database	2
SQLite DataTypes	4
Creating Tables	4
Tutorial 2: Create a Table.....	5
sqlite3_exec()	5
Tutorial 3: Insert and Select Records	7
Selecting Data	8
Tutorial 4: Update Records	13
Tutorial 6: Delete Records	14
Assignment Submission.....	14

Time required: 90 minutes

- Comment each line of code as show in the tutorials and other code examples.
- Follow all directions carefully and accurately.
- Think of the directions as minimum requirements.

SQLite and C++

SQLite3 is the most-used SQL implementation in the world. It is a self-contained C library included in everything from portable devices to web browsers.

SQLite is a self-contained, file-based SQL database. SQLite does not come with C++. We will install a C library to work with SQLite.

Setup the Project: SQLite with C++

We need a C library to work with SQLite database. Download the latest sqlite-amalgamation(versionnumber).zip file from:

<https://www.sqlite.org/download.html>

Extract the contents of the ZIP file into your project folder. Only store this program in this folder. The compile batch will attempt to compile every file in the folder.

Tutorial 1: Connect to an SQLite Database

Use your SQL database you created in SQL Basics.

Please keep copies of each tutorial as you go in case something goes wrong. Use the name shown in the tutorial screenshot.

1. In your project folder, create a C++ file named: **Connect.cpp**
2. Insert the following code to create and/or connect to a database.

```

1  /**
2   * Filename: Connect.cpp
3   * Written by:
4   * Written on:
5   * Description: Connect to or create an SQLite3 database
6   */
7
8  #include <iostream>
9  #include "sqlite3.h"
10
11  Codiumate: Options | Test this function
12  int main()
13  {
14      // Pointer to SQLite connection
15      sqlite3 *db;
16
17      // Save the connection result
18      int dbOpen = 0;
19
20      // Save the result of opening the file
21      // If there is a database file, open it,
22      // otherwise create a new database file
23      dbOpen = sqlite3_open("data.db", &db);
24
25      // Was there an error opening the database file?
26      if (dbOpen)
27      {
28          std::cout << " DB Open Error: " << sqlite3_errmsg(db) << std::endl;
29      }
30      else
31      {
32          std::cout << " Database Opened Successfully!" << std::endl;
33      }
34
35      // Close the connection
36      sqlite3_close(db);
37      return (0);
38  }

```

SQLite3 is an external library. To compile it correctly, it must also be linked to the project.

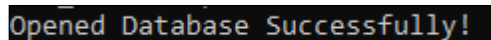
There are multiple files in this project, including the sqlite3 library files.

3. Create a batch file named **ConnectCompile.bat** with the following code.

```
rem Compile the sqlite3 c library. This only needs to be done once.
rem Place rem in front of this line to not compile it each time.
gcc sqlite3.c -c
rem This line compiles the sqlite3 library and the project file.
g++ Connect.cpp -o Connect.exe sqlite3.o -lsqlite3
pause
```

Run the batch file to compile the program.

Example run:

A terminal window with a black background and green text displaying the message "Opened Database Successfully!".

4. You should see **data.db** in your project folder.

SQLite DataTypes

- **NULL:** The value is a NULL value.
- **INTEGER:** Store a whole number.
- **REAL:** Floating-point value, for example, 3.14, the value of PI.
- **TEXT:** A text string. TEXT value stored using UTF-8, UTF-16BE or UTF-16LE encoding.
- **BLOB:** The value is a blob of data, i.e., binary data. It is used to store images and files.

The following Python types convert to SQLite types.

C++ Types	SQLite Types
None	NULL
int	INTEGER
float	REAL
std::string	TEXT
bytes	BLOB

Creating Tables

In an SQL database, data is stored in tables. Tables define a set of columns and fields, much like a spreadsheet. They contain 0 or more rows with data for each of the defined fields.

Let's create a table named **products** that tracks the following data. This is a data dictionary.

prod_id (primary key)	INTEGER
prod_name	TEXT
prod_desc	TEXT
prod_price	REAL
prod_qty	INTEGER

We will create a **products** table using SQLite3 in C++.

SQL is a scripting language like Python. You can assign SQL code to a string variable.

Tutorial 2: Create a Table

Working with SQLite3 includes three main functions: `sqlite3_open()`, `sqlite3_exec()`, and `sqlite3_close()`.

In the previous part, the functions `sqlite3open()` and `sqlite3_close()` were used to open and close the connection.

To execute SQL, the function `sqlite3_exec()` is used.

sqlite3_exec()

Calls to `sqlite3_exec()` use five parameters:

- Database connection (SQLite3 pointer)
- SQL to run
- Callback function
- First argument to callback function
- Address of where to write error messages

Because SQL commands like `SELECT` can return multiple results, the callback function is used to act on them. The fourth parameter is optional when working with functions that may not need it.

To create a table within the existing database, the `CREATE TABLE` keywords are used in SQL.

Let's create a table.

```

1  /**
2   * Filename: CreateTable.cpp
3   * Written by:
4   * Written on:
5   * Description: Create a SQLite3 table
6   */
7
8  #include <iostream>
9  #include <string>
10 #include "sqlite3.h"
11 // Function prototype for callback function
12 int callback(void *NotUsed, int argc, char **argv, char **azColName);
13
14 Codiumate: Options | Test this function
15 int main()
16 {
17     // Pointer to SQLite connection
18     sqlite3 *db;
19     // Save any error messages
20     char *zErrMsg = 0;
21     // Save the connection result
22     int dbOpen = 0;
23     // Variable to store SQL commands
24     std::string sql;

```

```

25     // ----- OPEN DATABASE FILE -----//
26     // Save the result of opening the file
27     // If there is a database file, open it,
28     // otherwise create a new database file
29     dbOpen = sqlite3_open("data.db", &db);
30
31     // Was there an error opening the database file?
32     if (dbOpen)
33     {
34         // Show an error message
35         std::cout << "DB Error: " << sqlite3_errmsg(db) << std::endl;
36         // Close the connection
37         sqlite3_close(db);
38         // Return an error
39         return (1);
40     }
41     std::cout << " Database Opened Successfully!" << std::endl;

```

```

43 // ----- CREATE TABLE -----//
44 // SQL string to create a table
45 sql = "CREATE TABLE IF NOT EXISTS PRODUCTS ("
46     "prod_id INTEGER PRIMARY KEY NOT NULL,"
47     "prod_name TEXT NOT NULL,"
48     "prod_desc TEXT,"
49     "prod_price REAL,"
50     "prod_qty INTEGER)";
51
52 // Run the SQL (convert the string to a C-String with c_str() )
53 dbOpen = sqlite3_exec(db, sql.c_str(), callback, 0, &zErrMsg);
54
55 std::cout << " Product Table Created!" << std::endl;
56
57 // Close the SQL connection
58 sqlite3_close(db);
59 return 0;
60 }
61
62 // Callback function for SQL operations
63 Codiumate: Options | Test this function
64 int callback(void *NotUsed, int argc, char **argv, char **azColName)
65 {
66     // Return successful
67     return 0;
68 }

```

Example run:

```

Opened Database Successfully!
Product Table Created!

```

Tutorial 3: Insert and Select Records

We have a created a table. This is the SQL for inserting records

```

"INSERT INTO PRODUCTS VALUES (1, 'Minecraft', 'Building game', 15.99, 1);
"INSERT INTO PRODUCTS VALUES (2, 'Tetris', 'Sort blocks', 12.99, 23);"
"INSERT INTO PRODUCTS VALUES (3, 'Pong', 'Paddle wars', 2.99, 6);"
"INSERT INTO PRODUCTS VALUES (4, 'Guessing Game', 'What number?', .99, 200);"

```

- INSERT INTO PRODUCTS specifies that we are inserting into the PRODUCTS table.
- VALUES says that the values are coming up next.

- The values must match the datatypes of the table fields.

```
sql = "INSERT INTO PRODUCTS VALUES "  
      "(1, 'Minecraft', 'Building game', 15.99, 1);";
```

Selecting Data

Selecting and then fetching the data from records is simple as inserting them. The execute method uses the SQL command of getting all the data from the table using "**SELECT * from table_name**". The results are accessed with a loop.

The callback function takes four parameters:

- void *NotUsed: Not used. This can be passed from the sqlite3_exec() function directly.
- int argc: Number of results
- char **argv: Array of values
- char **azColName: Array of column names


```

1  /**
2   * Filename: InsertRecords.cpp
3   * Written by:
4   * Written on:
5   * Description: Delete, create table, insert and select records
6   */
7
8  #include <iostream>
9  #include <string>
10 #include "sqlite3.h"
11 // Function prototype for callback function
12 int callback(void *NotUsed, int argc, char **argv, char **azColName);
13
14 Codiumate: Options | Test this function
15 int main()
16 {
17     // Pointer to SQLite connection
18     sqlite3 *db;
19     // Save any error messages
20     char *zErrMsg = 0;
21     // Save the result of opening the file
22     int dbOpen;
23     // Save any SQL
24     std::string sql;

```

```

25 // ----- OPEN DATABASE FILE -----//
26 // Save the result of opening the file
27 // If there is a database file, open it,
28 // otherwise create a new database file
29 dbOpen = sqlite3_open("data.db", &db);
30
31 if (dbOpen)
32 {
33     // Show an error message
34     std::cout << "DB Error: " << sqlite3_errmsg(db) << std::endl;
35     // Close the connection
36     sqlite3_close(db);
37     // Return an error
38     return (1);
39 }
40 std::cout << " Database Opened Successfully!" << std::endl;
41
42 // ----- DROP TABLE -----//
43 sql = "DROP TABLE IF EXISTS products";
44
45 // Run the SQL (convert the string to a C-String with c_str() )
46 dbOpen = sqlite3_exec(db, sql.c_str(), callback, 0, &zErrMsg);
47 std::cout << " Products Table Dropped!" << std::endl;

```

```

49 // ----- CREATE TABLE -----//
50 // SQL string to create a table
51 sql = "CREATE TABLE IF NOT EXISTS products ("
52     "prod_id INT PRIMARY KEY NOT NULL,"
53     "prod_name    TEXT    NOT NULL,"
54     "prod_desc    TEXT,"
55     "prod_price   REAL,"
56     "prod_qty     INTEGER);";
57
58 // Run the SQL (convert the string to a C-String with c_str() )
59 dbOpen = sqlite3_exec(db, sql.c_str(), callback, 0, &zErrMsg);
60 std::cout << " Products Table Created!" << std::endl;
61
62 // ----- INSERT -----//
63 // Save SQL insert data
64 sql = "INSERT INTO products VALUES "
65     "(1, 'Minecraft', 'Building game', 15.99, 1);"
66     "INSERT INTO products VALUES "
67     "(2, 'Tetris', 'Sort blocks', 12.99, 23);"
68     "INSERT INTO products VALUES "
69     "(3, 'Pong', 'Paddle wars', 2.99, 6);"
70     "INSERT INTO PRODUCTS VALUES "
71     "(4, 'Guessing Game', 'What number?', .99, 200);";
72
73 // Run the SQL (convert the string to a C-String with c_str() )
74 dbOpen = sqlite3_exec(db, sql.c_str(), callback, 0, &zErrMsg);
75 std::cout << " Records inserted!" << std::endl;

```

```

77 // ----- SELECT -----//
78 // Save SQL insert data
79 sql = "SELECT * FROM products";
80
81 // Run the SQL (convert the string to a C-String with c_str() )
82 dbOpen = sqlite3_exec(db, sql.c_str(), callback, 0, &zErrMsg);
83
84 // Close the SQL connection
85 sqlite3_close(db);
86 return 0;
87 }

```

```

89 // callback function for SQL operations
    Codiumate: Options | Test this function
90 int callback(void *NotUsed, int argc, char **argv, char **azColName)
91 {
92     // int argc: holds the number of results
93     // (array) azColName: holds each column returned
94     // (array) argv: holds each value
95
96     for (int i = 0; i < argc; i++)
97     {
98         // Show column name, value, and newline
99         std::cout << " " << azColName[i] << ": "
100         | << argv[i] << std::endl;
101     }
102     // Insert a newline
103     std::cout << std::endl;
104     // Return successful
105     return 0;
106 }

```

Example run:

```

Database Opened Successfully!
Products Table Dropped!
Products Table Created!
PROD_ID: 1
PROD_NAME: Minecraft
PROD_DESC: Building game
PROD_PRICE: 15.99
PROD_QTY: 1

PROD_ID: 2
PROD_NAME: Tetris
PROD_DESC: Sort blocks
PROD_PRICE: 12.99
PROD_QTY: 23

PROD_ID: 3
PROD_NAME: Pong
PROD_DESC: Paddle wars
PROD_PRICE: 2.99
PROD_QTY: 6

PROD_ID: 4
PROD_NAME: Guessing Game
PROD_DESC: What number?
PROD_PRICE: 0.99
PROD_QTY: 200

```

Tutorial 4: Update Records

Rows in an SQLite database can be modified using UPDATE SQL statement. We can update single columns as well as multiple columns.

This is the general form to update existing table data.

```
UPDATE TABLE_NAME SET column1 = value1, column2 = value2 WHERE condition;
```

In the above syntax, the SET statement is used to set new values to the column, the WHERE clause is used to select the rows for which the columns are needed to be updated.

Minecraft has a price change. We can change Minecraft's row in the PRODUCT table to reflect this change:

1. We issue an UPDATE SQL statement to change the PROD_PRICE of Minecraft to its new value of 19.99.
2. The WHERE clause in the UPDATE statement ensures we only change the value of PROD_PRICE if a row has PROD_ID = 1.

Add the following code to update our products data. Notice that we are running 2 SQL statements in a single command.

```
84      // ----- UPDATE -----//
85      sql = "UPDATE products SET prod_price = 19.99 WHERE prod_id = 1;"
86      |    "SELECT * FROM products;";
87      dbOpen = sqlite3_exec(db, sql.c_str(), callback, 0, &zErrMsg);
88
89      // Close the SQL connection
90      sqlite3_close(db);
91      return 0;
92  }
```

Example run. The example only display the changed record.

```
PROD_ID: 1
PROD_NAME: Minecraft
PROD_DESC: Building game
PROD_PRICE: 19.99
PROD_QTY: 1
```

Tutorial 6: Delete Records

Removing data from a table is the same as inserting in SQL. Using DELETE FROM and the table name, data can be removed based on conditions.

```
DELETE FROM TABLE_NAME [WHERE Clause]
```

The project is almost done. Add the following code to finish the project.

```
88      // ----- DELETE -----//
89      sql = "DELETE FROM products WHERE prod_id = 4;"
90          "SELECT * FROM products;";
91      dbOpen = sqlite3_exec(db, sql.c_str(), callback, 0, &zErrMsg);
92
93      // Close the SQL connection
94      sqlite3_close(db);
95      return 0;
96  }
```

Example run:

Record 4 will be deleted.

Assignment Submission

1. Attach the program files.
2. Attach screenshots showing the successful operation of the program.
3. Submit in Blackboard.