Basics of SQL using SQLite

by Phairoj Jatanachai

What is SQL?

- SQL stands for Structured Query Language.
- It is a programming language designed for managing and manipulating relational databases.
- Two common pronunciations for SQL:
  - "S-Q-L" (pronouncing each letter individually)
  - "Sequel":
    This pronunciation originated with the early development of SQL at IBM, where it was initially called "SEQUEL" (Structured English Query Language).

How SQL Works

- SQL operates on tables, which consist of rows and columns.
- Each table holds structured data like a spreadsheet.
- Example table:

id	name	email	age
1	Alice	alice@example.com	25
2	Bob	bob@example.com	30
3	Charlie	charlie@example.com	35

Create (Insert data into the table)

```
CREATE TABLE users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    email TEXT UNIQUE NOT NULL,
    age INTEGER
);
```

id	name	email	age
1	Alice	alice@example.com	25
2	Bob	bob@example.com	30
3	Charlie	charlie@example.com	35

```
INSERT INTO users (name, email, age) VALUES ('Alice', 'alice@example.com', 25);
INSERT INTO users (name, email, age) VALUES ('Bob', 'bob@example.com', 30);
INSERT INTO users (name, email, age) VALUES ('Charlie', 'charlie@example.com', 35);
```

Read (Retrieve data from the table)

```
-- Retrieve all data
SELECT * FROM users;

-- Retrieve specific columns
SELECT name, age FROM users;

-- Retrieve filtered data (users older than 30)
SELECT * FROM users WHERE age > 30;

-- Sort data by age in descending order
SELECT * FROM users ORDER BY age DESC;
```

Update (Modify existing data)

```
-- Update Alice's age
UPDATE users SET age = 26 WHERE name = 'Alice';

-- Update Bob's email
UPDATE users SET email = 'bob.new@example.com' WHERE name = 'Bob';
```

Delete (Remove data from the table)

```
-- Delete Bob from the table
DELETE FROM users WHERE name = 'Bob';


-- Delete all users older than 30
DELETE FROM users WHERE age > 30;
```

Advances Quries

```
SELECT * FROM users WHERE age > 30;

SELECT * FROM users ORDER BY age DESC;

SELECT COUNT(*) FROM users;

SELECT AVG(age) AS average_age FROM users;

SELECT age, COUNT(*) AS count FROM users GROUP BY age;
```

Summary of CRUD Operations

Operation	SQL Command
Create	CREATE TABLE table_name (...);
Insert	INSERT INTO table_name (...) VALUES (...);
Read	SELECT ... FROM table_name WHERE ...;
Update	UPDATE table_name SET ... WHERE ...;
Delete	DELETE FROM table_name WHERE ...;

SQLite

- QLite is a lightweight, self-contained SQL database engine. It's easy to use for beginners and doesn't require a server to run.
- It's commonly used for mobile apps, small applications, or for learning SQL.

Setting up SQLite

1. Install SQLite:
   - macOS/Linux: SQLite is often pre-installed. Open a terminal and type:
     sqlite3 –version
     If it's not installed, use your package manager, e.g., brew install sqlite (for macOS) or sudo apt install sqlite3 (for Linux).
   - Windows: Download the precompiled binaries from SQLite Download Page and add them to your system path.
2. Launch SQLite:
   sqlite3 my_database.db
   This creates a new database file named my_database.db.
3. Quit:
   .quit

Using SQLite with C++

1. Install SQLite
   - Install SQLite
   - Include the SQLite library in your project:
     - Compile SQLite into a static or shared library, or
     - Use the precompiled SQLite library (sqlite3.h and sqlite3.c).
2. Basic SQLite Workflow in C++
   - The general workflow is:
     1) Open or create a database.
     2) Execute SQL commands.
     3) Handle the results.
     4) Close the database.
3. Compile:

```
g++ -std=c++11 -o sqlite_example  sqlite_example.cpp -lsqlite3
```

```cpp
#include <iostream>
#include <sqlite3.h>

int main() {
    sqlite3* db;                    // Database connection
    char* errMsg = nullptr;         // Error message
    int rc;                         // Return code

    // Open or create a database file
    rc = sqlite3_open("my_db.db", &db);
    if (rc) {
        std::cerr << "Can't open database: " << sqlite3_errmsg(db) << std::endl;
        return rc;
    } else {
        std::cout << "Opened database successfully!" << std::endl;
    }
```

Pointer to database connection object

Database file name

Open an existing SQLite database file.
Create a new SQLite database file if it does not already exist.

```
// Create a table
    const char* createTableSQL = R"(
        CREATE TABLE IF NOT EXISTS users (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            name TEXT NOT NULL,
            email TEXT NOT NULL,
            age INTEGER NOT NULL
        );
    )";

    rc = sqlite3_exec(db, createTableSQL, nullptr, nullptr, &errMsg);
    if (rc != SQLITE_OK) {
        std::cerr << "SQL error: " << errMsg << std::endl;
        sqlite3_free(errMsg);
    } else {
        std::cout << "Table created successfully!" << std::endl;
    }
```

A raw string literal in C++.
Define a multi-line string in C++ without requiring escape characters

SQLITE_OK (0): The command executed successfully.
Non-zero error codes indicate a failure (e.g., SQLITE_ERROR, SQLITE_BUSY, SQLITE_MISUSE).

This is an optional argument passed to the callback function.

Callback Function: Used for SQL commands that return rows (like SELECT).

```
// Insert data into the table
   const char* insertSQL = R"(
      INSERT INTO users (name, email, age) VALUES ('Mac', 'susans@yahoo.com', 95);
      INSERT INTO users (name, email, age) VALUES ('David', 'bob@hotmail.com', 30);
   )";

   rc = sqlite3_exec(db, insertSQL, nullptr, nullptr, &errMsg);
   if (rc != SQLITE_OK) {
      std::cerr << "SQL error: " << errMsg << std::endl;
      sqlite3_free(errMsg);
   } else {
      std::cout << "Records inserted successfully!" << std::endl;
   }
```

The name of the i-th column (e.g., "id", "name").

```
// Read data from the table
    const char* selectSQL = "SELECT * FROM users;";
    auto callback = [](void* NotUsed, int argc, char** argv, char** azColName) -> int {
        for (int i = 0; i < argc; i++) {
            std::cout << azColName[i] << ": " << (argv[i] ? argv[i] : "NULL") << std::endl;
        }
        std::cout << std::endl;
        return 0;
    };

    rc = sqlite3_exec(db, selectSQL, callback, nullptr, &errMsg);
    if (rc != SQLITE_OK) {
        std::cerr << "SQL error: " << errMsg << std::endl;
        sqlite3_free(errMsg);
    } else {
        std::cout << "Data retrieved successfully!" << std::endl;
    }

    // Close the database connection
    sqlite3_close(db);
    return 0;
```

id	name	age
1	Alice	25
2	Bob	30

For the first row:
- argv[0] = "1"
- argv[1] = "Alice"
- argv[2] = "25"

For the second row:
- argv[0] = "2"
- argv[1] = "Bob"
- argv[2] = "30"

FAQ