Report: Using SQLite in C++ with Visual Studio

1. Introduction

SQLite is a lightweight, self-contained, and serverless database engine. It is widely used for desktop and mobile applications because it requires no separate server process. This report explains how to set up SQLite in **Microsoft Visual Studio**, integrate it into a C++ project, and use it with sample code.

2. Tools & Requirements

- Microsoft Visual Studio (any recent version, e.g., 2019/2022).
- SQLite source code (`sqlite3.c` and `sqlite3.h`) downloaded from the official site.
- A new C++ Console Application.

3. Setup Instructions

Option A (Recommended: Easier Way – Compile SQLite Directly into Project)

1. Create a new **Console App (C++)** in Visual Studio.
2. Copy `sqlite3.c` into the **Source Files** folder of your project.
3. Copy `sqlite3.h` into the **Header Files** folder.
4. In your C++ file (`main.cpp`), include the SQLite header:

```
#include "sqlite3.h"
```

This way, you don't need `sqlite3.lib` or `sqlite3.dll`. SQLite will compile directly into your project.

4. Sample Code

main.cpp

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

using namespace std;

// Callback function to print query results
static int callback(void* data, int argc, char** argv,
```

```
char**azColName) {
    cout << (const char*)data << endl;
    for (int i = 0; i < argc; i++) {
        cout << azColName[i] << " = " << (argv[i] ? argv[i] :
"NULL") << endl;
    }
    cout << "------------------" << endl;
    return 0;
}

int main() {
    sqlite3* DB;
    char* messageError;
    int exit = sqlite3_open("student.db", &DB);

    if (exit) {
        cerr << "Error opening database: " << sqlite3_errmsg(DB) <<
endl;
        return -1;
    }
    cout << "Opened database successfully!" << endl;

    // Create table
    string sql = "CREATE TABLE IF NOT EXISTS STUDENTS("
                 "ID INTEGER PRIMARY KEY AUTOINCREMENT, "
                 "NAME TEXT NOT NULL, "
                 "AGE INT NOT NULL);";

    exit = sqlite3_exec(DB, sql.c_str(), NULL, 0, &messageError);
    if (exit != SQLITE_OK) {
        cerr << "Error Create Table: " << messageError << endl;
        sqlite3_free(messageError);
    } else {
        cout << "Table created successfully!" << endl;
    }

    // Insert data
    sql = "INSERT INTO STUDENTS (NAME, AGE) VALUES('Laiba', 21);"
          "INSERT INTO STUDENTS (NAME, AGE) VALUES('Khalil', 22);";

    exit = sqlite3_exec(DB, sql.c_str(), NULL, 0, &messageError);
    if (exit != SQLITE_OK) {
```

```
        cerr << "Error Insert: " << messageError << endl;
        sqlite3_free(messageError);
    } else {
        cout << "Records inserted successfully!" << endl;
    }

    // Select data
    sql = "SELECT * FROM STUDENTS;";
    cout << "\nFetching Data..." << endl;
    exit = sqlite3_exec(DB, sql.c_str(), callback, (void*)"Result
Row:", &messageError);
    if (exit != SQLITE_OK) {
        cerr << "Error Select: " << messageError << endl;
        sqlite3_free(messageError);
    }

    sqlite3_close(DB);
    return 0;
}
```

5. Expected Output

When you run the program, you will see something like:

```
Opened database successfully!
Table created successfully!
Records inserted successfully!

Fetching Data...
Result Row:ID = 1
NAME = Laiba
AGE = 21
------------------
Result Row:
ID = 2
NAME = Khalil
AGE = 22
------------------
```

6. Conclusion

- SQLite is very easy to integrate into a C++ project.
- By directly compiling `sqlite3.c` into the project, you don’t need `.lib` or `.dll` files.
- With only two files (`sqlite3.c` and `sqlite3.h`), you can create, insert, and query data.