Sqlp - Simple terminal query and .sql file processing for Sqlite3

Pablo Edronkin, 2019

https://orcid.org/0000-0001-8690-7030

Abstract.

Sqlp is a very small program written in C++ that allows the user to send SQL queries to a Sqlite3 database from a common terminal without the use of any application or library other than itself and Sqlite3.

Keywords.

Sqlite, sqlp, sqlite3, data, database, sql

Acknowledgements.

This project could not have taken place without the work done by:

- The developers of Sqlite3[6.1.].

License and info for contributors.

Please read the following files included with this project:

- README.md: contains info on setting up your system and installing the files related to this module.

- CONTRIBUTING.md: if you want to contribute to this project.

- COPYING: for license information.

Table of Contents

1. Introduction.

Sqlite3 is one of the most widely used relational database systems; it is very simple, lightweight and fast, and while it cannot be compared with great server - based relational database systems in terms of versatility or capabilities, it does fill a niche for many users, probably most of those that need a relational database system without the complications involved in maintaining a server – based alternative such as PostgreSQL, Mysql or MariaDB.

Recently I have been developing a dataset based on a Sqlite3 database. Being a relational system, datasets constructed using Sqlite3 are far better to handle and to safely keep data than common flat file formats used for dataset creation, no t to mention the fact that data scientists usually spend more time preparing the data from a conventional dataset than actually using the data to do the actual research.

In this regard, SQL offers several advantages over tools used for flat files, plus, data integrity is better on relational databases. However, Sqlite3 has some limitations, and one of those is that it is not easy to execute a significant number of queries or SQL sentences at once using scripts or programs, like it is generally possible with products like those I mentioned before.

The development of this particular dataset requires constant updates and a processing protocol for the data entered that soon became repetitive, so I decided to write myself a program to help me automate the whole work, and thus I wrote sqlc.

This is not an ambitious project but a home – grown solution, so to speak. It does not pretend to be a super tool of any kind but it does the work of executing independent SQL queries or series of several queries grouped into a single file.

2. Conventions, caveats and base conditions for development.

Take into consideration these factors.

2.1. Terms and abbreviations used.

- Composite query: a query passed as argument to sqlc that is in fact, a succession of two or more queries separated one from the other by a “;” character.

- DB: Data base.

- RDB: Relational data base.

- RDBMS: Relational data base management system.

- .sql file: a flat file that contains various SQL queries. Such a file contains, in fact, a composite query.

2.2. Caveats.

While using sqlc you should be aware of:

2.3.1. SQL dialect.

Sqlite3 uses its own SQL dialect. This program does not do anything with the code passed to Sqlite3; however you still need to pass adequate code. If it seems that Sqlc does nothing when you are attempting to use it, the most likely reason is that your SQL has an error, as understood by Sqlite3. In that case:

2.3.1.1. Check that your SQL is syntactically correct, according to Sqlite3. SQL code can be tricky, especially in the case of complex queries.

2.3.1.2. Check that you did not include any commented text, blank lines or odd characters.

2.3.1.3. In the case of composite queries and .sql files, check that you have placed a ";" between each query and at the end of the last one.

2.3.1.4. Test each query on its own before building composite queries or .sql files.

2.3.2. Retrieval of query results.

Some queries return results, while others do not. A SELECT produces essentially a selection or listing, while an INSERT does not.

In the case of those SQL queries that do return results in the form of listings, sqlp stores them into a file called *sqlp_results.txt*, in which the values for every field and record are separated with the character "|".

Note from example **[2.3.3.2]** below that in the case of a composite query, results for the second one will be appended to the results of the first query in *sqlp_results.txt*. It is up to the user in such a case to parse those results adequately, or to perform separate calls to sqlp doing a results – producing query on each case, and extracting the data from *sqlp_results.txt* on each occasion.

Keep in mind that each time that sqlc is called, it overwrites sqlp_results.txt, but on each call, if results are retrieved more than once, listings are appended at the end of sqlp_results.txt, thus:

2.3.2.1. sqlp_results.txt with one listing

*./sqlc your_database “SELECT * FROM your_table_1;”*

will overwrite any existing instance of sqlp_results.txt in the working folder and fill it with the contents of the SELECT operation described above.

2.3.2.2. sqlp_results.txt with two or more listings

*./sqlc your_database “SELECT * FROM your_table_1; SELECT * FROM your_table_2;”*

Will also overwrite any existing instance of sqlp_results.txt and will fill it with the results of the first SELECT, and then with the results of the second query appended to the first.

2.3.3. Sql file and composite query syntax.

Sqlc accepts single and composite queries, as well as sql files as input. In all cases you should end each query with a “;” to tell Sqlite3 that each is a separate query.

Examples:

2.3.3.1. Single query

*“SELECT * FROM your_table_1;”*

2.3.3.2. Composite query

*“SELECT * FROM your_table_1; SELECT * FROM your_table_2;”*

2.3.4. System calls.

In theory, Sqlp can be called via a system call from any program written in any language without the need of having language – specific libraries installed. It can also be called directly from the terminal prompt.

If , when making a call, sqlp answers you with

“Incorrect number of arguments”

You should:

2.3.4.1. First check that you have in fact written the number of arguments required by the program.

2.3.4.2. Check that the strings corresponding to each argument have no spaces between characters. This might be interpreted by sqlp as additional arguments and hence, the program rejects your call. In this case you might solve the problem by using quotes (“ “) to surround each one of your arguments.

2.3.4.3. Sqlite3 also requires quotes (‘ ‘ or “ “) for strings. In this case, surround your query with double quotes (“ “), and the strings inside each query with single quotes (‘ ‘).

2.3.4.4. If the problem remains and you are sure about the number of arguments and the correctness of the SQL code, you might need to tell Sqlite3 that you are passing in fact, the characters that correspond to quotes (\‘ \’ instead of ‘ ‘ and \“ \” instead of “ “).

This happens, for example, when you call sqlp from within the system function in Scheme, so you would have to write something like:

*(system "./sqlp \"your_database\" \"SELECT * FROM your_table_1 WHERE your_string = \'Hello world!\';\" ")*

While the same call from within a terminal would be:

./sqlp your_database *"SELECT * FROM your_table_1 WHERE your_string = \'Hello world!\';"*

or even:

*./sqlp your_database "SELECT * FROM your_table_1 WHERE your_string = 'Hello world!';"*

Other languages might require a different syntax, so you might have to try different combinations.

3. Development.

This is a very simple, short program written in C++. I wrote it as a time – saving tool to use during the development of a dataset that required some degree of data preparation using a somewhat lengthy and repetitive process.

It is not meant to do miracles, but to pass SQL queries to Sqlite3 using just two arguments, which are the database name and the string representing the query or file that contains the queries.

In the second case, when sqlp is called with an .sql file as second argument, sqlp opens the file and converts all its contents into a single string, that then is passed to Sqlite3. Once data is retrieved from the .sql file and converted into a rather long string, sqlp treats that data exactly as in the case of a short string query.

3.1. Possible developments in the future.

Some ideas that might work in the future, but I have not tested yet are:

3.1.1. Embed sqlp in a multi threaded environment.

Sqlp could work in a multi threaded environment provided by OpenMP or some MPI library integrated into a program that calls sqlp. The only thing to consider is how table locking will be managed by Sqlite3 in practice.

3.1.2. Call sqlp from within an uncommon language.

It would be possible to use sqlp with a language like Scheme or Fortran using a system call and thus eliminating the need for Sqlite3 libraries for those languages that are hard to come by and not updated often.

3.1.3. Provide multi threading capability from within sqlp.

MPI would probably be overkill, but OpenMP could feasibly work. The issue with table locking remains and SQL scripts for a multi threaded environment would require some extra care, but it is still possible.

3.2. Operating system.

I developed sqlp on a Linux machine to use within a Linux environment with gcc. I see no problems for any reasonably seasoned user to compile it for a Windows system.

4. Structure

In the main folder of the project you will find a couple of .cpp and .hpp files, as well as text files containing license and contribution information.

The main project folder also contains two sub folders:

4.1. /doc

This folder contains the manual for sqlp.

4.2. /examples

This folder contains a fe example files, including a databases and .sql files.

5. Use

5.1. Installation

Uncompress and copy the contents of this file wherever you want in your system.

Copy *sqlite3.cpp* and *sqlite3.h* to the same folder or open *sqlp1.hpp* and modify the *#include "sqlite3.h"* line in *sqlp1.hpp* accordingly, with the path to *sqlite3.h* on your system.

5.1.1. Compilation

In a Linux system using gcc:

g++ -std=c++17 -Wall -O3 -c sqlp0.cpp sqlp1.cpp

Link as:

g++ -std=c++17 -Wall -O3 sqlp0.o sqlp1.o -o sqlp -lsqlite3

Users with other operating systems or compilers will find little or no trouble adapting the compile and link snippets above. Please let me know your results in this regard.

5.2. Deinstallation

Delete the folder containing the contents of this file from your system.

5.3. Calling sqlp

Assuming that you have sqlp on the same folder that holds your database and you are using a Linux system, briefly, in a terminal write:

./sqlp [db] [query]

where

[db] is the database that you want to query.

[query] is a Sqlite3 - compatible SQL query or the name of a .sql file.

For example:

*./sqlp your_database "SELECT * FROM your_table_1;"* will send this SELECT query to *your_database* and return the data of all records contained in *your_table_1*. The results will be saved to *sql_results.txt*.

*./sqlp your_database "SELECT * FROM your_table_1;SELECT * FROM your_table_2;"* will send both SELECT queries (composite query) to *your_database*, and *sql_results.txt* will store the results of both queries.

./sqlp your_database your_file.sql will send the queries contained in *your_file.sql* to your_database as a composite query. See the /*examples* folder for more.

Query results are written as a continuous string to *sqlp_results.txt*. Data fields are separated with a "|" character.

Pay attention to the path to *[db]* and *[query]* if it is a .sql file. If they are not in the same folder as sqlc, then you will need to provide full paths or add those files or folders to your $PATH.

6. Sources

6.1. Sqlite.org. (2000). SQLite Home Page. [online] Available at: https://www.sqlite.org/index.html [Accessed 26 Aug. 2019].

Alphabetical Index

F

G

H

I

L

M

O

P

Q

R

S

T

W

Y