Sqlp - Simple terminal query and .sql file processing for Sqlite3 and HDF5 databases.

Pablo Edronkin, 2019 - 2020

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 or HDF5 database from a common terminal or shell.

It is not meant as a substitution of Sqlite3's dot commands or HDF5 database editors, but rather as a complement. In some cases, using those commands and programs might prove more convenient, while in others, using sqlp may have its own advantages.

Keywords.

Sqlite, sqlp, sqlite3, data, database, sql, hdf5, hdfql

Acknowledgements.

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

- The developers of Sqlite3[6.1.].

- The developers of HDF5[6.3.].

- The developers of HDFql[6.4.].

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.

Also read the documentation regarding Sqlite3, HDF5 and HDFql licenses which should be taken into account in addition to the license applicable to sqlp.

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, not 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. I also needed something simpler than the command line interpreter[6.2.] that characterizes Sqlite3. So I decided to write myself a program to help me automate the whole work, and thus I wrote sqlp.

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.

Sqlite3 has some very good tools to achieve similar tasks as sqlp, but alt least in some cases, the simpler syntax of sql could make it more convenient: dot commands are interpreted by Sqlite’3’s command line interpreter, and not Sqlite3 itself, while sqlp passes everything directly to Sqlite3, no questions asked.

After the initial phase of development involving Sqlite3 compatibility, I began studying the feasibility of using sqlp with other formats. HDF5 and the fairly newly – developed HDF query language (HDFql) soon proved an interesting venue to pursue for various reasons, including the

practicality of HDF5 for storing large amounts of data, similitudes between SQL and HDFql, and the adaptable nature of sqlp's code.

HDFql, like Sqlite3, comes with a command line interpreter and its queries are quite similar to those of the latter, so I decided to adapt it to sqlp. As a result, the main advantage in this regard, at least in my view, is that sqlp offer a single user interface both for Sqlite's version of SQL and HDFql, which is a SQL – like query language for HDF5 databases.

Much like in the case of Sqlite, for now support for HDF5 via HDFql has some limitations in sqlp, but it essentially works.

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 sqlp that is in fact, a succession of two or more queries separated one from the other by a semicolon (“;”) 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.

- .hql file: a flat file that contains various HDFql queries similar in all aspects to .sql files except for the fact that they operate with HDFql statements on HDF5 databases.

- .db: Sqlite3 database file.

- .h5: HDF5 database file.

- Field divisor character: pipe character (“|”) used to separate items or fields in a file used to display results of a query.

2.2. Caveats.

While using sqlp you should be aware of:

2.3.1. SQL dialect for Sqlite3.

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 sqlp 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 semicolon 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 and HDFql 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 field divisor character **[2.1.]**.

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 sqlp 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

*./sqlp your_database.db "SELECT * FROM your_table_1;"*

for SQL queries, or

./sqlp your_database.h5 "SELECT FROM your_dataset_1"

for HDFql queries.

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

*./sqlp your_database.db "SELECT * FROM your_table_1; SELECT * FROM your_table_2;"*

for Sqlite3 or:

./sqlp your_database.h5 “SELECT FROM your_dataset_1; SELECT FROM your_dataset_2;”

for HDF5 *w*ill 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, HDFql files and composite query syntax.

Sqlp accepts single and composite queries, as well as sql files as input. In all cases you should end each query with a semicolon 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;”*

Note that sqlp provides a similar procedure for HDFql so that composite queries and query files (.hql) are also possible.

This is a superset provided by sqlp in order to homogenize the behavior of both database system options and thus make it easier to handle them. Neither HDF5 or HDFql provide to my knowledge, something similar at the time of this writing.

HDFql composite and .hdf syntax is similar to that of composite SQL and .sql files. The only difference is that you should write only HDFql queries for HDF5 databases and Sql queries only for Sqlite3 databases.

Do not mix SQL and HDFql statements in the same composite queries or files.

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 single and/or double 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 or HDFql code, you might need to tell Sqlite3 or the HDF5 libraries 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. Note that HDFql queries are written very similarly to those equivalent in SQL, and the same caveats regarding special characters apply.

2.3.5. Differences between SQL and HDFql.

The creators of HDFql did an excellent work in providing HDF5 systems with a query language that is similar to SQL because it eases quite considerably the learning curve involved in learning to interact with HDF5 files using its native API for C, C++, etc.

However, partly because HDFql is fairly new, and partly because HDF5 is not a relational system, HDFql and SQL have some ostensibly simple but really significant differences at this point. However, the points raised above are largely applicable to HDFql as well.

When designing sqlp I tried to provide a standard interface for both, meaning that sending queries to Sqlite3 or HDF5 databases would be as similar as possible.

In practice this meant that I had to write some code to make some slight differences between both disappear. The most significant of these adaptation is the possibility to use query files and

composite queries for HDF5: While these possibilities are native to Sqlite3, they are not in the case of HDF5, neither in the case of HDFql.

In this version of sqlp parsing composite and .sql file queries in the case of Sqlite3 is left to Sqlite3 itself in order to stick to that standard, while .hql and HDFql composite queries are handled by means of an emulator embedded into sqlp.

3. Development.

This is a rather 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 and HDFql queries to HDF5 databases 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.

An embedded emulator does the same for .hql files and HDFql composite queries. This means that the user only has to learn one interface for both database types.

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 for Sqlite3 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.

At this moment I am using non – MPI libraries for HDF5 and HDFql, but compilation does make use of OpenMP.

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.

HDF5 support for Fortran is better than in the case of relational databases.

3.1.3. Provide multi threading capability from within sqlp.

MPI would probably be overkill and complicate things for the user quite abit, 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.

HDF5 files are better prepared, on the other hand, to deal with multi – threading since they have been conceived from the beginning to store very large amounts of data, something that more often than not requires very fast access.

Please take into account that currently, sqlp does not employ MPI for HDF5 access.

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, except whenever it might be employed within a cluster environment using any for of MPI. MS Windows systems will probably be unable to handle MPI calls.

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 few 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 -fopenmp -I[path to your HDFql include folder] [path to your HDFql wrapper/cpp folder]/libHDFql.a -ldl

where

[path to your HDFql include folder]

is the path on your computer to the include folder of HDFql. In my case, for example, this is

/usr/include/hdfql/serial/include/

and for the wrappers

/usr/include/hdfql/serial/wrapper/cpp/

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. In order to uninstall OpenMP, Sqlite3 and other libraries involved, refer to their respective manuals.

5.3. Calling sqlp with two arguments.

Assuming that you have sqlp on the same folder that holds your database, you are using a Linux system, and you do not want to use a macro, briefly, in a terminal write:

./sqlp [db] [query]

where

[db] is the database that you want to query, either a .db or .h5 type.

[query] is a Sqlite3 - compatible SQL query or the name of a .sql file if performed against a .db file or an HDFql query performed against a .h5 database.

For example:

*./sqlp your_database.db "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.db "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.db 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. At the end of the query results you will find *"|EOQ|"* for "End Of Query", meaning that one query – for example, a SELECT – produced a listing that reaches op to that point.

This is useful whenever you run multiple queries that produce results. With *"|EOQ|"* you could parse *sql_query_results.txt* in a more easy way to separate the results that correspond to one query from those from another one.

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

5.4. Calling sqlp with three arguments.

If you want to use macros with sqlp, the syntax for a call uses one more argument, which is the macro that you want to use, like:

./sqlp [db/file] [query] [macro]

Where [macro] represents the macro name that you intend to call. Currently there are these macros:

5.4.1. OPEN_QUERY_CLOSE

This macro will open the database, perform the query you want, and then close the database. In the case of queries that produce listings as results, they will be available at file sqlp_results.txt. You will need a text editor to see them or use OPEN_QUERY_CLOSE_SHOW to see them no your console.

Remember that data in sqlp_results.txt is not formatted; it is just a string of characters and it is up to the user – be it a program or a person - to make sense of those contents.

5.4.2. TEST_DB

This macro will not execute a query, but will test if the database is available or not. If the database does not exist, it will create it, since Sqlite3 does that and sqlp passes every argument to Sqlite3 as is.

To change the behavior of TEST_DB you would need to change Sqlite3's behavior by means of reconfiguration or custom programming.

5.4.3. OPEN_QUERY_CLOSE_SHOW

This one does the same as OPEN_QUERY_CLOSE but after finishing, shows the contents of sqlp_results.txt. It is important to remember that the contents of this file are not formatted, so in the case of complex queries you may see a lot of pretty confusing characters if you don't know what you are looking for.

Yet, for simple queries it might be convenient, as well as for testing queries and evaluating the kind of results that they produce.

5.4.4. PRETTY_SHOW

This query acts not on a database, be it Sqlite3 or HDF5 but on the results provided by queries made upon those databases. What PRETTY_SHOW does is to format the results contained in the file sqlp_results.txt in a columnar structure so that they appear legible to the human eye.

Instead of the name of a database, the user has to provide the name of the file to read and turn pretty – usually *sqlp_results.txt* as

"[file_name]"

Instead of an SQL or HDFql query, the user has to provide the number of columns that will be used to format the results as a string of the following form:

"COLS=[number_of_columns];"

Notice the semicolon at the end of the string.

And the third argument should be the macro:

"PRETTY_SHOW"

Thus, the complete call to sqlp in order to use PRETTY_SHOW would be something like this in a hypothetical case of results organized in five columns, based on

./sqlp "sqlp_results.txt" "COLS=5;" "PRETTY_SHOW"

As a result of the use of this macro, a file called *sqlp_pretty_tmp.txt* will be created. By opening this file you would see the results of the last query in a somewhat legible way.

In effect, when you call sqlp using two arguments and no macro, the program by default will execute with OPEN_QUERY_CLOSE. In the future other macros might be designed.

6. Sources

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

6.2. Sqlite.org. (2000). *Command Line Shell For SQLite*. [online] Available at: https://www.sqlite.org/cli.html [Accessed 10 Oct. 2019].

6.3. The HDF Group. (2019). *The HDF Group - ensuring long-term access and usability of HDF data and supporting users of HDF technologies*. [online] Available at: https://www.hdfgroup.org [Accessed 18 Oct. 2019].

6.4. Hdfql.com. (2019). *The easy way to manage HDF5 data*. [online] Available at: http://www.hdfql.com/ [Accessed 18 Oct. 2019].

7. Alphabetical Index

M

O

P