Ali Al Kadhim and the FSU HGHCAL Team
May, 2022

Queries and sqlite

sqlite3 Basics

Before anything, of course make sure you have the repository cloned by doing

```
git clone https://github.com/AliAlkadhim/FSU_HGCAL_DB.git
```

sqlite is a full-featured relational database system. Sqlite does not use a client-server architecture: all the code for the database is contained in the driver . All the data for each database is contained in a simple *platform-independent* file. It is powerful, secure, and especially efficient since it doesn't use a server. It is included in most python distributions, so it was preinstalled on 619. If your python distribution for some reason doesn't include sqlite3, install it with the command:

```
sudo apt-get install sqlite3
```

for Debian architectures. If you have Mac, then you already have it preinstalledthen it comes pre-installed on Macsudo apt-get install pandoc texlive-latex-base texlive-fonts-recommended texlive-extra-utils texlive-latex-extra sudo apt-get install pandoc texlive-latex-base texlive-fonts-recommended texlive-extra-utils texlive-latex-extra ! It is both an executable and a python module, so tunning `sqlite3` in the terminal will open up the sqlite3 shell, such that you can run any of the commands in the shell directly. Alternatively, you can use it as a python module as I shall describe below.

All these tutorials are also available in PDF format in the `docs/pdfs` directory of this repository (this repository is https://github.com/AliAlkadhim/FSU_HGCAL_DB).

sqlitebrowser

Even more convenient, sqlite databases (that is, databses ending in `.db`) allow one to use sqlitebrowser to do quick inspection, queries and other commands using a Graphical User Interface (GUI), without much knowledge of any syntax. If you don't have sqlitebrowser, **I highly recommend that you install it** and use it. If you have Debian architectures, it can be installed with

```
sudo apt-get install sqlitebrowser
```

If you have Windows, download the appropriate installer from `https://sqlitebrowser.org/dl/`

If you have MacOS, install it with the command

```
brew install --cask db-browser-for-sqlite
```

or by downloading the latest release on `https://sqlitebrowser.org/dl/`.

After you have installed sqlitebrowser, you can open our database `TESTME.db`, which is located in the `tests` directory with the command

```
sqlitebrowser TESTME.db
```

This can be done by anyone at home! go to the `Browse Data` tab, and choose which table you want from the `Table` dropdown menu. After choosing a table from the dropdown, type any query you would like under the column(s) that you would like to use (i.e. type under the columns where it says "Filter").

Using sqlite3 for queries (and everything else) on the command line

In the terminal, go to the `tests` directory of this repository, and open up our database by doing

```
sqlite3 TESTME.db
```

Now, to show a list of the tables that are available in this database, do

```
.tables
```.

You can examine each table in this list. For now we have the following tables: 1) FULL_SENSOR_LOGISTICS
2) MOS_GCD_LOGISTICS
3) STRIP_SENSOR_LOGISTICS
4) DIODES_NP_LOGISTICS
5) PQC_LOGISTICS
6) HPK_STRUCTURES_LOGISTICS 7) IV_GRADING_RESULTS

The most useful command for queries is the `SELECT` command. Suppose you want to see everything in the `STRIP_SENSOR_LOGISTICS` table, then do the command

```
SELECT * FROM STRIP_SENSOR_LOGISTICS;
```

This displays everything from this table (SQL is not case-sensetive, so it works if you don't capetalize the above command. However, we capitalize SQL commands by convention to distinguish them from non-SQL syntax. Also, yes, the semicolon at the end of the command is necessary). Also, sql supports tab-comletion of course.

Notice that everything in the output is squeeszed together. This is because the default display ("output") mode is "list". You can double check this by doing `.mode`. This will output "current output mode: list". The nice thing is that there are may display modes you can choose from:

Available display modes:

- ascii
- box
- csv
- column
- html
- insert
- json
- line
- list
- markdown
- quote
- table
- tabs
- tcl

You can examine these modes, and see which ones you like. I like the "markdown" mode, because it displays all the column names with large spacing. I really recommend switching the mode. You can switch to it by doing

```
.mode markdown
```

Suppose you want to do the same query as above but order the output by the Sensor_ID, then simply do

```
SELECT * FROM STRIP_SENSOR_LOGISTICS ORDER BY Sensor_ID;
```

You can even sort them in ascending or descending order by adding `ASC` or `DESC` to the query above.

Suppose you want to know about the sensor with the Sensor_ID N4788_7, then you can display all its information from this table by doing

```
SELECT * FROM STRIP_SENSOR_LOGISTICS WHERE Sensor_ID ='N4788_7';
```

Say you want to inspect the IV grading results, then the table is IV_GRADING_RESULTS and for this the best mode is list (can be implemented by doing `.mode list`) since there are so many columns). Suppose you're interested in seeing whether the sensor with Sensor_ID N4791_2 has passed the various tests, then just do

```
SELECT * FROM IV_GRADING_RESULTS where Sensor_ID='N4791_2';
```

Suppose you want to see which sensors (idenfified by Sensor_ID) have passed the "FOUR__More_than_two_neighbour_cells_bad__requirem_ONE_and_TWO_HPK_FULL_PROBE_CARD" requirement, then do

```
SELECT Sensor_ID, FOUR__More_than_two_neighbour_cells_bad__requirem_ONE_and_TWO_HPK_FULL_PROBE_CARD FROM IV_GRADING_RESULTS
```

Or if you want to look at them ordered by Sensor_ID, then do

```
SELECT Sensor_ID, FOUR__More_than_two_neighbour_cells_bad__requirem_ONE_and_TWO_HPK_FULL_PROBE_CARD FROM IV_GRADING_RESULTS ORDER BY Sensor_ID DESC;
```

Better yet, since "FOUR__More_than_two_neighbour_cells_bad__requirem_ONE_and_TWO_HPK_FULL_PROBE_CARD" is a very lengthy name for a column, you can display it as some other name of your choosing. For example, you could display it as "REQUIREMENT_FOUR_HPK" by doing

```
SELECT Sensor_ID, FOUR__More_than_two_neighbour_cells_bad__requirem_ONE_and_TWO_HPK_FULL_PROBE_CARD AS REQUIREMENT_FOUR_HPK FROM IV_GRADING_RESULTS ORDER BY Sensor_ID DESC;
```

Suppose you want to look at all the sensors that passed both the "FOUR__More_than_two_neighbour_cells_bad__requirem_ONE_and_TWO_HPK_FULL_PROBE_CARD" criteria and the "THREE__More_than_8_bad_cells__requirem_ONE_and_2__CMS_FULL_PROBE_CARD_OFF_DF" criteria, then do

```
SELECT Sensor_ID, FOUR__More_than_two_neighbour_cells_bad__requirem_ONE_and_TWO_HPK_FULL_PROBE_CARD FROM
```

`IV_GRADING_RESULTS WHERE FOUR__More_than_two_neighbour_cells_bad__requirem_ONE_and_TWO_HPK_FULL_PROBE_CARD='Passed' AND THREE__More_than_8_bad_cells__requirem_ONE_and_2__CMS_FULL_PROBE_CARD_OFF_DF='Passed' ORDER BY Sensor_ID DESC;`

There is basically limitless types of quick queries you can do. If you would like to learn more about all the different things and queries you can do with sqlite, I highly recommend visiting the sqlite documentation https://www.sqlite.org/cli.html#:~:text=Start%20the%20sqlite3%20program%20by,name%20will%20be%20created%20automatically.

Once you're done with sqlite, just do

`.exit`.

Queries with sqlite in python

All such queries described above can be done by anyone using the sqlite3 module in python. First import the sqlite3 modeule.

```
try:
    import sqlite3 as sql
    have_sqlite3 = True
except ImportError:
    sqlite3=None
    print("you don't have sqlite3, do 'sudo apt-get install sqlite3'")
    ```

Now do any query directly in python.
```

def query(query_string):

```
connection=None
try:
    DB_NAME = '../TESTME.db'
    connection = sql.connect(DB_NAME)
    cursor = connection.cursor()

except sqlite3.Error as err:
    print('could not open database %s' % err)
    if connection:
        connection.rollback()
    exit(1)
cur = cursor.execute(query_string)
col_names = [cn[0] for cn in cur.description]
print(col_names,'\n')
rows = cur.fetchall()
print(rows)


if connection:
    connection.close()
```

```
For example,
```

query_string ="""SELECT * FROM IV_GRADING_RESULTS WHERE Sensor_ID='N4788_5'"""

query(query_string) ```
```