Data Modelling Tools

AUTM08016

Topic 3
Build an SQLite Database

Dr Diarmuid Ó Briain
Version 1.0 [01 January 2024]

TUS
Ollscoil Teicneolaíochta na Sionainne:
Lár Tíre, An tIarthar Láir
Technological University of the Shannon:
Midlands Midwest

Dr Diarmuid Ó Briain

Linux Version

```
~$ lsb_release -a | grep Description
Description:   Ubuntu 22.04.3 LTS
```

VirtualBox version

```
~$ virtualboxvm --help | head -1
Oracle VM VirtualBox VM Runner v7.0.12
```

SQLite versions

```
~$ sqlite3 --version
3.37.2 2022-01-06 13:25:41
872ba256cbf61d9290b571c0e6d82a20c224ca3ad82971edc46b29818d5dalt1

~$ sqlitebrowser --version
DB Browser for SQLite Version 3.12.1
Qt Version 5.15.1
SQLite Version 3.33.0.
```

Table of Contents

Table of Figures

1. SQLite Database

1.1 Introduction

Databases provide a structure for the organised collection of information that can be easily accessed, managed and updated. Python can access many database types and the examples here use the SQLite form of database.

- **SQLite library**: is a Relational DataBase Management System (RDBMS) contained in a C library. In contrast to many other RDBMS, SQLite is not a client–server database engine, it is embedded into the end program and therefore is a popular choice as embedded database software for local/client storage.
- SQLite is **Atomicity, Consistency, Isolation, Durability (ACID)** compliant and implements most of the SQL standard, generally following PostgreSQL syntax.
- **sqlite3**: is a terminal-based front-end to the SQLite library that can evaluate queries interactively and display the results in multiple formats. **sqlite3** can also be used within shell scripts and other applications to provide batch processing features.
- **DB Browser for SQLite**: GUI editor for SQLite databases.

1.2 Objectives

At the end of this topic the learner will be able to

- Build an SQLite database
- Perform SQL queries on the database
- Review the data within the database
- Build a database using the Python sqlite3 module

2. Installation on GNU/Linux

Install **SQLite3**, the **DB Browser for SQLite** and the **Python module** to interface with the SQLite library.

```
~$ sudo apt install -y sqlite3 sqlitebrowser
```

3. Structured Query Language (SQL)

SQL is a specific language used in programming and designed for managing data held in a RDBMS, or for stream processing in a relational data stream management system. In this case, the RDBMS is SQLite.

Using the **sqlite3** terminal client open a new database.

```
~$ sqlite3 db_1.sqlite
```

Check if there are any current tables in the database.

```
sqlite> .tables
```

Create a table called **class_list**, note that each field must be defined. The **CREATE TABLE** query, a fundamental SQL statement, is used to define the structure of a new table within a database. It specifies the table's name, the data types of each column, and any additional constraints or restrictions that govern the data stored within the table.

3.1 Create a table

```
sqlite> CREATE TABLE IF NOT EXISTS class_list (ref_no INTEGER
PRIMARY KEY, fname TEXT, sname TEXT, number INTEGER);
```

- **CREATE TABLE IF NOT EXISTS**: This clause ensures that the table is created only if it doesn't already exist. This prevents errors if you accidentally attempt to create a duplicate table.
- **class_list**: This is the name of the table being created.
- **ref_no INTEGER PRIMARY KEY**: This defines a column named **ref_no** with data type **INTEGER** and sets it as the table's primary key. The primary key uniquely identifies each row in the table.
- **fname TEXT**: This defines a column named **fname** with data type **TEXT**, which can store character strings of any length.
- **sname TEXT**: This defines a column named **sname** with data type **TEXT** for storing character strings of any length.
- **number INTEGER**: This defines a column named **number** with data type **INTEGER** for storing numeric values.

Confirm the table has been created.

```
sqlite> .tables
class_list
```

3.2 Insert data into a database

Insert data into the new table.

```
sqlite> INSERT INTO class_list (ref_no, fname, sname, number)
VALUES (0, 'Tom', 'Ryan', 111111);
```

- **INSERT INTO**: This keyword indicates the beginning of the **INSERT** statement.
- **class_list**: This specifies the name of the table where the data will be inserted.
- **(ref_no,fname,sname,number)**: This defines the column names where the data will be inserted. The parentheses indicate the start of a column list.
- **VALUES (0,'Tom','Ryan',111111)**: This specifies the values to be inserted into the columns. The parentheses indicate the start of a value list.

Repeat the process for other data.

```
sqlite> INSERT INTO class_list (ref_no, fname, sname, number)
VALUES (1, 'Mary', 'Murphy', 222222);
sqlite> INSERT INTO class_list (ref_no, fname, sname, number)
VALUES (2, 'Ada', 'Lovelace', 333333);
sqlite> INSERT INTO class_list (ref_no, fname, sname, number)
VALUES (3, 'Charles', 'Babbage', 444444);
```

Confirm the data is in the table via the terminal program.

```
sqlite> SELECT * FROM class_list;
0|Tom|Ryan|111111
1|Mary|Murphy|222222
2|Ada|Lovelace|333333
3|Charles|Babbage|444444
```

Open the database using the **`sqlitebrowser`**.

```
ada:~$ sqlitebrowser db_1.sqlite
```

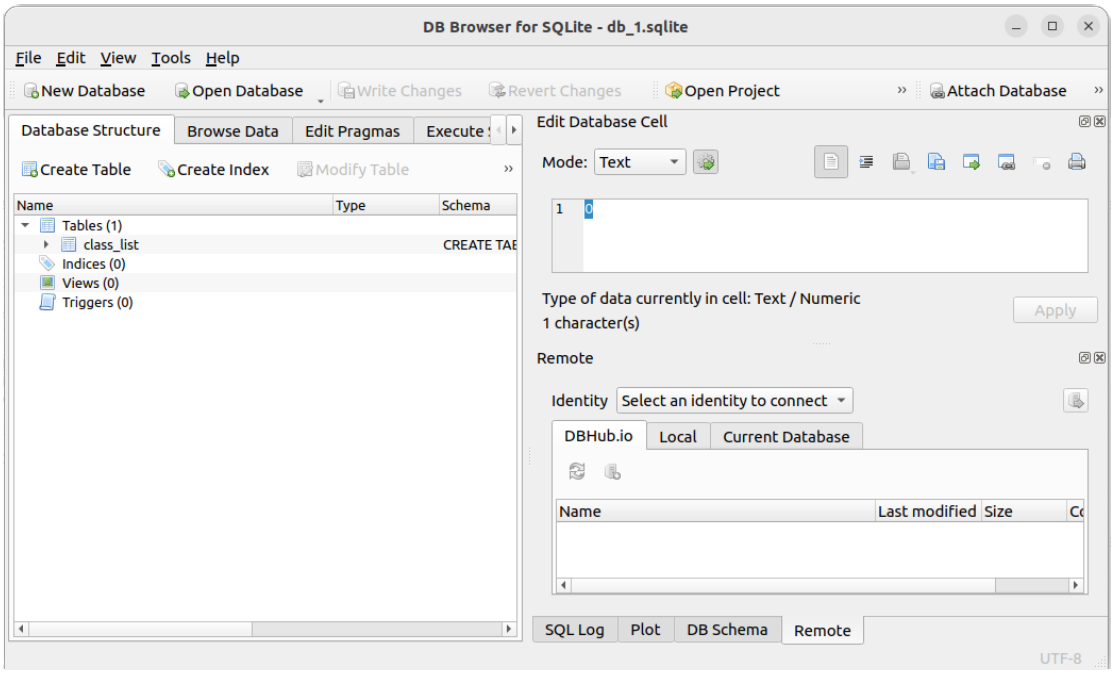

Figure 1: sqlitebrowser

Confirm the data is in the table using the **`sqlitebrowser`** viewer functionality.

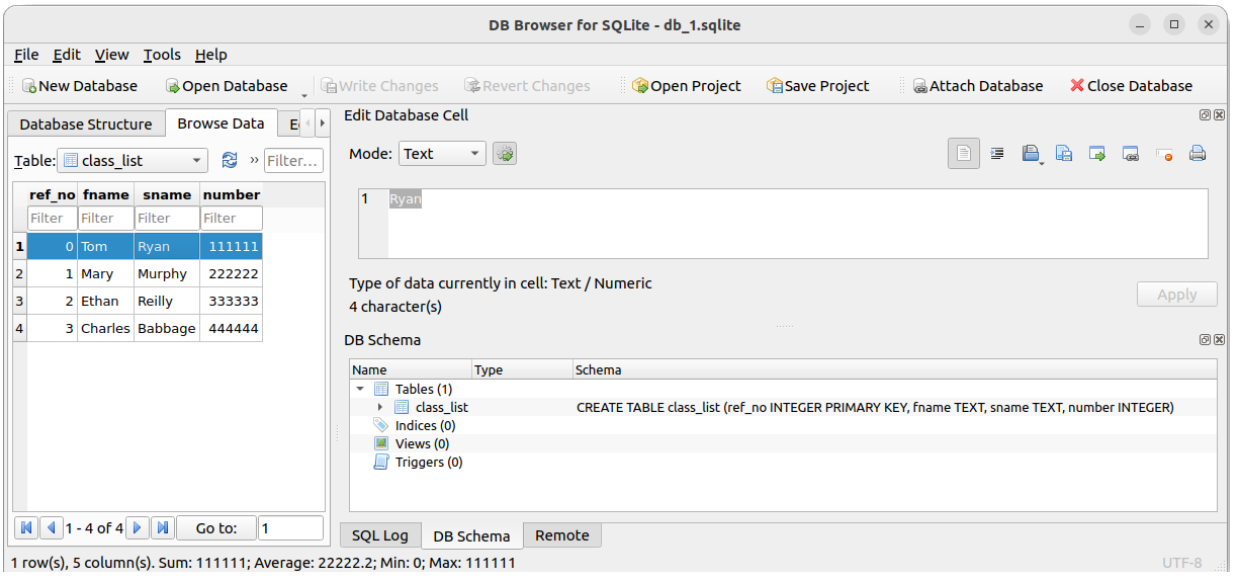

Figure 2: Use the sqlitebrowser to access the data

3.3 Update data in the database

Update a piece of data, in this case the name of the person in line **2** with the number **333333**. The SQL query updates the first name of all students in the **class_list** table whose contact number matches the pattern **333333**. The query uses the **LIKE** operator to search for values that partially match the specified pattern.

```
sqlite> UPDATE class_list SET fname = 'Ethan' WHERE number LIKE
333333;
```

- **UPDATE class_list**: This keyword indicates the beginning of the **UPDATE** statement.
- **SET fname = 'Ethan'**: This clause specifies the column to be updated and the new value to be assigned to it.
- **WHERE number LIKE 333333**: This clause specifies the condition that determines which rows will be updated. In this case, the query updates all rows where the value in the number column matches the pattern **333333**.

Carrying out a similar update, this time updating **sname** for the reference number is **2**.

```
sqlite> UPDATE class_list SET sname = 'Reilly' WHERE ref_no LIKE
2;
```

Confirm the change.

```
sqlite> SELECT * FROM class_list;
0|Tom|Ryan|111111
1|Mary|Murphy|222222
2|Ethan|Reilly|333333
3|Charles|Babbage|444444
```

3.4 Select from the Database using Wildcard

Search the database using the SELECT query. Here the search is in the **sname** field of the **class_list** table. Note that searching for **y** yielded nothing despite that letter existing in two family names. Why, well not name is just a **y** and the wildcard **%** is needed.

```
sqlite> SELECT * FROM class_list WHERE sname LIKE "y";
```

Here is an example employing the wildcard before **y**. Murphy is returned but not Ryan. This is because the wildcard is employed before the **y** and with none after the **y** only a name where **y** is the last letter can match.

```
sqlite> SELECT * FROM class_list WHERE sname LIKE "%y";
1|Mary|Murphy|222222
```

Employing the wildcard either side of the **y** in the query returns all names that include the letter **y**.

```
sqlite> SELECT * FROM class_list WHERE sname LIKE "%y%";
0|Tom|Ryan|111111
1|Mary|Murphy|222222
```

3.5 Delete data from the database

Delete some data from the table. The SQL query deletes all rows from the **class_list** table whose reference number starts with the digit **3**.

```
sqlite> DELETE FROM class_list WHERE ref_no LIKE 3;
```

- **DELETE FROM class_list**: This keyword indicates the beginning of the **DELETE** statement.
- **WHERE ref_no LIKE 3**: This clause specifies the condition that determines which rows will be deleted. The query deletes all rows where the value in the **ref_no** column starts with the digit **3**.

Confirm the change.

```
sqlite> SELECT * FROM class_list;
0|Tom|Ryan|111111
1|Mary|Murphy|222222
2|Ethan|Reilly|333333
```

Delete all the data from the table and confirm.

```
sqlite> DELETE FROM class_list;
sqlite> SELECT * FROM class_list;
```

Exit from the database terminal client.

```
sqlite> .quit
```

4. Exercise Laboratory #1

Write an SQL Database that includes information on some cars outside the window. If you cannot see cars make them up.

Include

1. Car manufacturers
2. Car model
3. Car colour
4. Car registration
5. Wheel type, alloy, etc...

Notes:

5. The python sqlite3 module

The example in Figure 3 demonstrates a connection to an **SQLite database**, the dropping and creation of a new table, the insertion of data and the reading (select) of data from the database. Carefully consider the code and follow what occurs when it is ran as demonstrated here.

The core of the SQL functionality is included in the function on lines 19 to 29. Start by creating a Connection object with **`sqlite3.connect(<db name>)`** that represents the database within the program (line 23). Next instantiate the **`Cursor` class** (line 24) which has the **`execute()`** method that permits the query to be sent to the database (line 25). Line 26 fetches all (remaining) rows of the query result, returning a list. This only results in values for SELECT queries and for all other query types an empty list is returned as no rows are available.

```
~$ cat python_sqlite.py
1  #! /usr/bin/env python3
2
3  import pprint
4  import sqlite3
5  import sys
6
7  # Defined variables
8  database = "db_2.sqlite"
9  table = "class_list"
10  columns = {
11      "ref_no": "INTEGER PRIMARY KEY",
12      "fname": "TEXT",
13      "sname": "TEXT",
14      "number": "INTEGER",
15  }
16  data = (
17      (0, "Tom", "Ryan", 111111),
18      (1, "Mary", "Murphy", 222222),
19      (2, "Ada", "Lovelace", 333333),
20      (3, "Charles", "Babbage", 444444),
21  )
22  list_ = list()
23  str_ = str()
24  tuple_ = tuple()
25
26  # // Query Function //
27  def query_(query):
28      """Query function for the Database"""
29
30      list_ = list()
31      with sqlite3.connect(database) as con:
32          cur = con.cursor()
33          cur.execute(query)
34          list_ = cur.fetchall()  # Empty except for SELECT
35          cur.close()  # Close database cursor
36          con.commit()  # Commit labels to the database
37      return (0, list_)
38
39
```

```python
# // Use SQL to drop a 'class_list' table if it currently exists //
print(f"Dropping '{table}' from the db '{database}' if it exists")
query_(f"DROP TABLE IF EXISTS {table}")

# // Use SQL to create new 'class_list' table //
print(f"Creating '{table}' in the '{database}' db")
for (key, value) in columns.items():
    list_.append(f"{key} {value}")
str_ = ", ".join(list_)
query_(f"CREATE TABLE IF NOT EXISTS {table} ({str_})")

# // Get input and put in the database table 'class_list' //
str_ = ", ".join(columns.keys())
for d in data:
    print(f"Inserting {d} into the '{table}' table")
    query_(f"INSERT INTO {table} ({str_}) VALUES {d}")

# // Getting data from database table 'class_list' //
print(f"Retrieving {d} from the '{table}' table")
(_, list_) = query_(f"SELECT * FROM {table}")

# // Printing table 'class_list' //
for t in list_:
    print("   ", ", ".join([str(e) for e in t]))

# End

```

Figure 3: python_sqlite.py

Run the program.

```
~$ ./python_sqlite.py
Dropping 'class_list' from the db 'db_2.sqlite' if it exists
Creating 'class_list' in the 'db_2.sqlite' db
Inserting (0, 'Tom', 'Ryan', 111111) into the 'class_list' table
Inserting (1, 'Mary', 'Murphy', 222222) into the 'class_list'
table
Inserting (2, 'Ada', 'Lovelace', 333333) into the 'class_list'
table
Inserting (3, 'Charles', 'Babbage', 444444) into the 'class_list'
table
Retrieving (3, 'Charles', 'Babbage', 444444) from the 'class_list'
table
    0, Tom, Ryan, 111111
    1, Mary, Murphy, 222222
    2, Ada, Lovelace, 333333
    3, Charles, Babbage, 444444
```

Confirm via the SQLite3 client terminal program and with the **`sqlitebrowser`**.

```
~$ sqlite3 db_2.sqlite
SQLite version 3.37.2 2022-01-06 13:25:41
Enter ".help" for usage hints.

sqlite> .tables
class_list

sqlite> SELECT * FROM class_list;
0|Tom|Ryan|111111
1|Mary|Murphy|222222
2|Ada|Lovelace|333333
3|Charles|Babbage|444444

sqlite> .quit
```

```
~$ sqlitebrowser db_2.sqlite
```

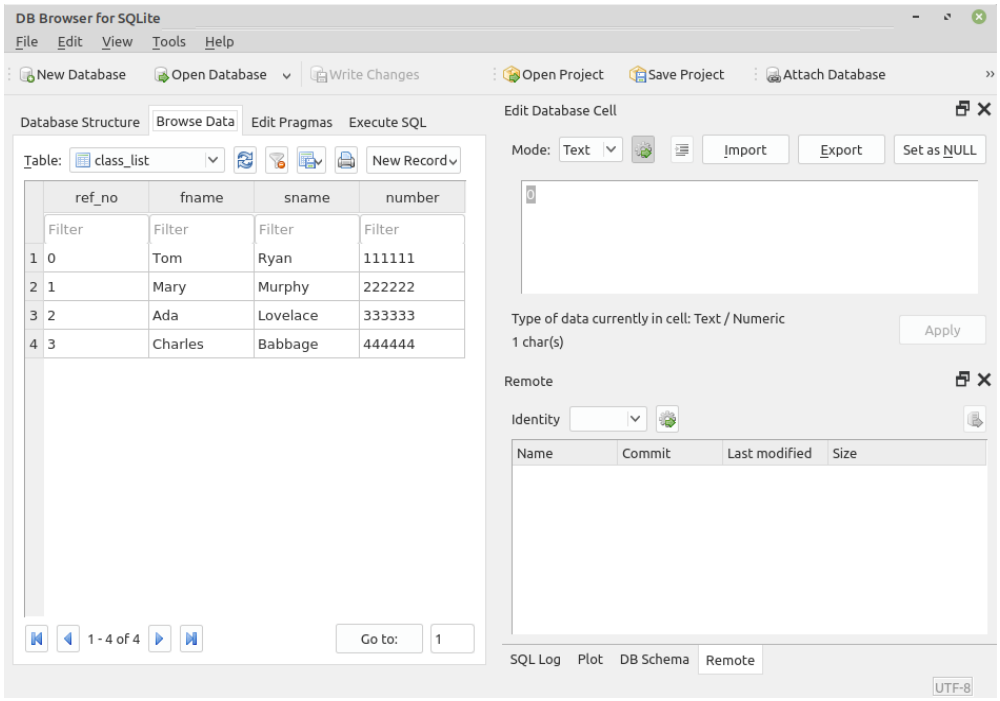

Illustration 1: BD Browser for SQLite #2

6. Exercise Laboratory #2

Write a program "**exercise_3.2.py**".

1. Add a shebang line and a document string "**Exercise #3.2 in Python3**".
2. Import the **sqlite3** module and declare global variables.
3. Copy the **query()** function from the "**python_dqlite.py**" program.
4. Open the database and retrieve the data from the "**class_list**" table.
5. Remove "**Mary Murphy**" from the course.
6. Replace "**Mary Murphy**" with her sister "**Nora**".
7. Add "**Leo Ashe**" to the course with a student number "**555555**".
8. Output the current state of the table.
9. Output should be like this:

```
~$ ./exercise_3.2.py
Retrieving data from the 'class_list' table
    0, Tom, Ryan, 111111
    1, Mary, Murphy, 222222
    2, Ada, Lovelace, 333333
    3, Charles, Babbage, 444444
Deleting 'Mary Murphy from the 'class_list'
Inserting 1, Nora, Murphy, 222222 into the 'class_list' table
Inserting 4, Leo, Ashe, 555555 into the 'class_list' table
Retrieving data from the 'class_list' table
    0, Tom, Ryan, 111111
    1, Nora, Murphy, 222222
    2, Ada, Lovelace, 333333
    3, Charles, Babbage, 444444
    4, Leo, Ashe, 555555
```

Notes: