**Book :** [**http://benaiahbooks.com/assets/upload/booklet/small.pdf**](http://benaiahbooks.com/assets/upload/booklet/small.pdf)

**Python + SQL**

**Intro to Section**

There are many databases out there. Each is different regarding how it works with Python. We’ll be using SQLite for this tutorial.

SQLite is a C library that provides a lightweight disk-based database that doesn’t require a separate server process and allows accessing the database using a nonstandard variant of the SQL query language. Some applications can use SQLite for internal data storage. It’s also possible to prototype an application using SQLite and then port the code to a larger database such as PostgreSQL or Oracle.

Best part about SQLite is that it comes pre-installed on many machines. Otherwise too, its easy to install on any system.The syntax is comparetively simple and helps in quick learning. Also sqlite3 module is already a part of Python3 standard library.

Here’s the module for python3 SQLite connection :

* <https://docs.python.org/3/library/sqlite3.html>
* <https://www.geeksforgeeks.org/sql-using-python-set-3-handling-large-data>

**Database For The Internet Of Things :** SQLite is popular choice for the database engine in cellphones, PDAs, MP3 players, set-top boxes, and other electronic gadgets. SQLite has a small code footprint, makes efficient use of memory, disk space, and disk bandwidth, is highly reliable, and requires no maintenance from a Database Administrator.

**Application File Format :** Rather than using fopen() to write XML, JSON, CSV, or some proprietary format into disk files used by your application, use an SQLite database. You'll avoid having to write and troubleshoot a parser, your data will be more easily accessible and cross-platform, and your updates will be transactional.

**Website Database :** Because it requires no configuration and stores information in ordinary disk files, SQLite is a popular choice as the database to back small to medium-sized websites.

**Stand-in For An Enterprise RDBMS :** SQLite is often used as a surrogate for an enterprise RDBMS for demonstration purposes or for testing. SQLite is fast and requires no setup, which takes a lot of the hassle out of testing and which makes demos perky and easy to launch.

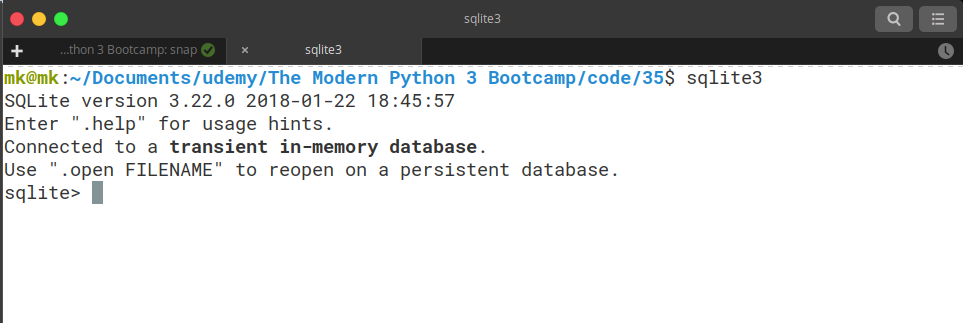
**Installing SQLite3**

Use the following command to install sqlite3.

**sudo apt install sqlite3**

**SQL Basics : Creating Tables**

Opening sqlite3 on terminal. It opens a transient **in-memory** database.

****

To exit the console you use :

**CTRL + D**

OR

**.quit()**

1. **.help** for showing all the options in sqlite3

2. **.tables**  for listing all the tables in sqlite3

**Creating a new table**

Storage Classes and Datatypes

1. NULL : value is a signed value  
2. INTEGER : signed integer values or 8 bytes

3. REAL : floating point value, 8-byte IEEE floating point number  
4. TEXT : The value is a text string, stored using the database encoding (UTF-8)

5. BLOB : The value is a blob of data

Reference : <https://www.sqlite.org/datatype3.html>

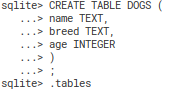
CREATE TABLE DOGS (

name TEXT,

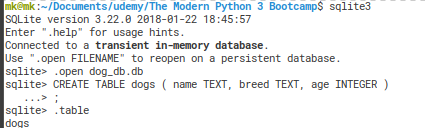
breed TEXT,

age INTEGER

);



The tables are lost once we close the terminal session. For making the tables persistent :



We use the **.open <database-name>** command before making a new table in the database.

This database is stored as a .db file in the same directory. When opening the database again, use the command :

> sqlite3 <database-name>

> sqlite3 dogs\_db.db

> .tables

dogs

If you want to work on a non-existent database, then you can use the same command above and a new database file will be created.

> sqlite3 cats\_db.db

**SQL Basics : Inserting**

We are going to see how to insert data into sqlite3 tables.

1. Insert Normal :

INSERT INTO CATS (NAME, BREED, AGE) VALUES (‘BLUE’,’SCOTTISHFOLD’,’3’)

SELECT \* FROM CATS

2. How to read sql from a file :

Write a sql file containing SQL commands. You can then read the sql file into the sqlite3 database using .read command

> .read basics.sql

> .tables

dogs

> SELECT \* FROM dogs;

Rose | Lab | 11

This is probably all about inserting.

There are two ways to import data, depending on the format of the data in the file to import. If the file is composed of SQL, you can use the .read command to execute the commands contained in the file.

If the file contains comma-separated values (CSV) or other delimited data, you can use the .import

[file][table] command. This command will parse the specified file and attempt to insert it into the

specified table. It does this by parsing each line in the file using the pipe character (|) as the delimiter and inserting the parsed columns into the table. Naturally, the number of parsed fields in the file should match up with the number of columns in the table. You can specify a different delimiter using the .separator command.

To see the current value set for the separator, use the .show command.

**SQL Basics : Selecting**

1. Select all columns from dogs:

Select \* from dogs

2. Select 2 columns from dogs:

Select name, age from dogs

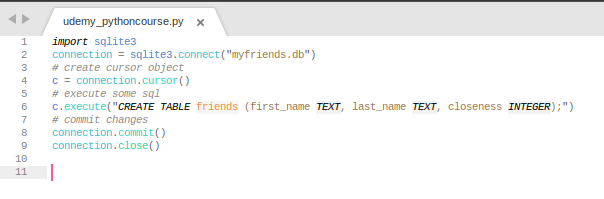
3. Filter

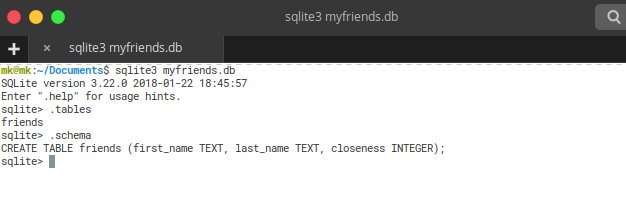
SELECT name FROM dogs WHERE name = ‘charlie’

SELECT \* FROM dogs WHERE breed IS NOT “Chihuahau” AND breed is NOT “Pug”;

**Connecting a DB with Python**

Python comes in with a built-in library for interacting with SQLite3. Its a connector b/w 2 technologies.

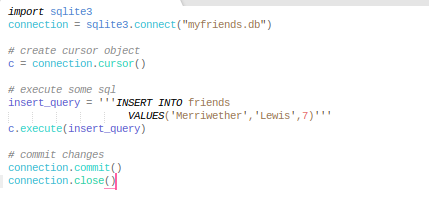


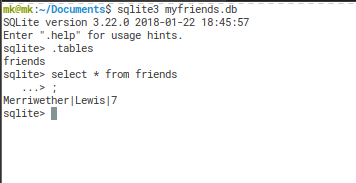


<https://stackoverflow.com/questions/6318126/why-do-you-need-to-create-a-cursor-when-querying-a-sqlite-database>

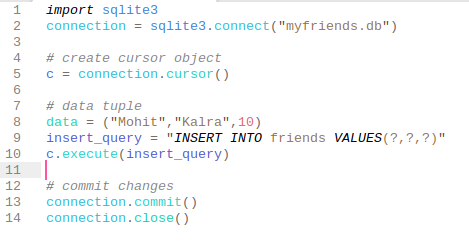
**Inserting with Python**

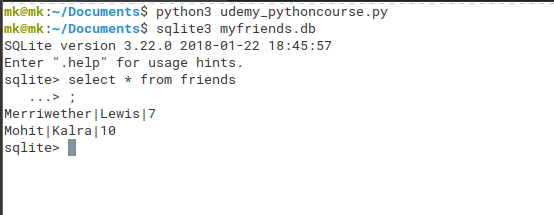
see that we use three quotes while we are breaking our query into the next line.

****

****

Another way of inserting data is using python tuples. We use ‘?’ instead of the actual value. This also helps in preventing SQL injection.





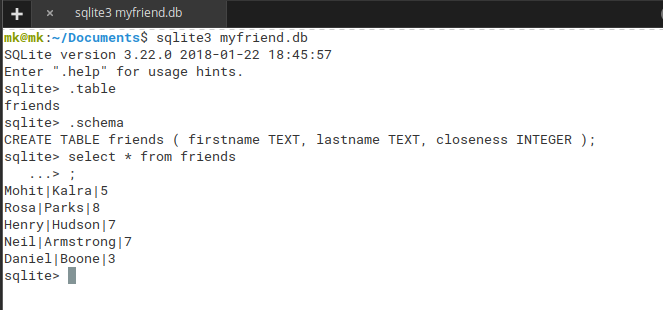
**Bulk Inserts with Python**

We are still working the same database myfriends.db. This part explores inserting data in bulk in Python.

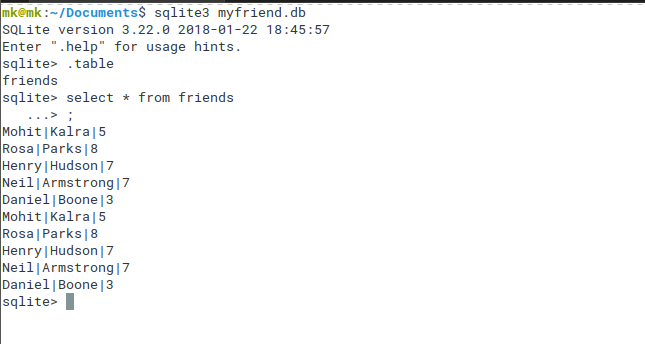
1. using **executemany()**

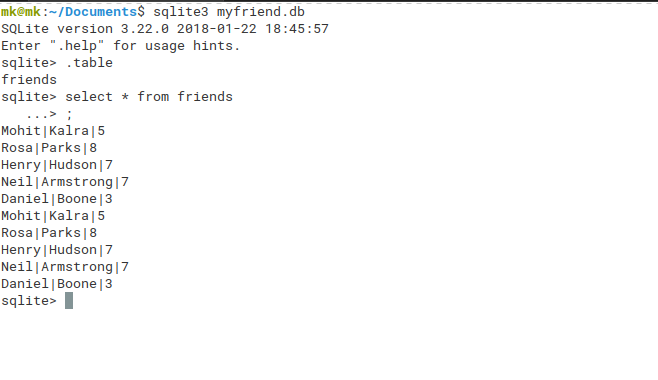


This inserts the above rows in the friends table in the ‘myfriend.db’ database.

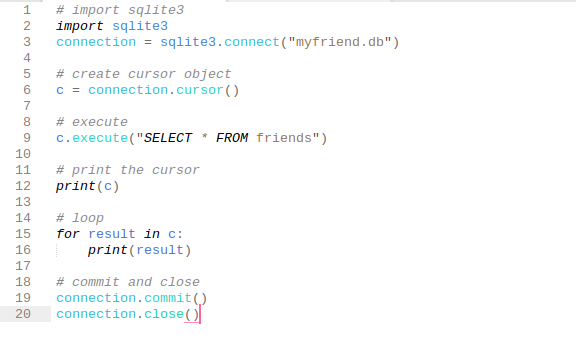


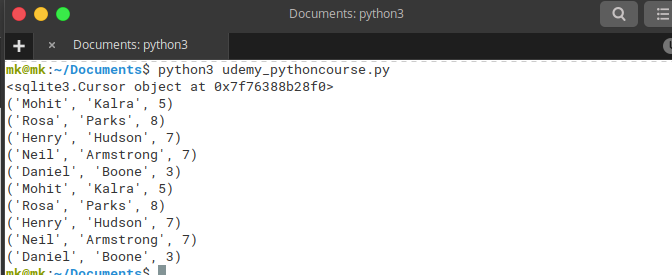
We could have also looped onto the list and used the execute() function as such. We can also calcualte the average closeness of the group of friends I am having.



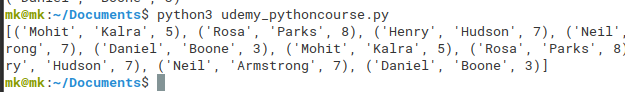


**Selecting with Python**

****

****

Another way is using cursor.fetchall() which returns the list of all results.



You can apply filters in the select query to make it more complex.

To fetch just one result, we can use the function **cursor.fetchone()**

A sample query:

SELECT \* FROM friends WHERE closeness > 5 ORDER BY closeness

In python we can use :

print(c.fetchall())

**SQL Injection** [**https://www.tutlane.com/tutorial/sqlite/sqlite-injection-attacks**](https://www.tutlane.com/tutorial/sqlite/sqlite-injection-attacks)

SQL injection is all about short-circuting the login sql query using OR 1=1 statement and using the sql comments in the SQL query “--”

This short circuits the SQL query and logs in the user.

To prevent this we used parametrized queries where we don’t just interpolate the queries but we pass it as a parameter which is sanitized before it is executed.

SQL statements can contain parameters. Parameters are placeholders in which values may be provided (or “bound”) at a later time after compilation. The following statements are examples of parameterized queries: insert into foods (id, name) values (?,?); insert into episodes (id, name) (:id, :name); These statements represent two forms of parameter binding: positional and named. The first command uses positional parameters, and the second command uses named parameters. Positional parameters are defined by the position of the question mark in the statement. The first question mark has position 1, the second 2, and so on. Named parameters use actual variable names, which are prefixed with a colon. When sqlite3\_prepare\_v2() compiles a statement with parameters, it allocates placeholders for the parameters in the resulting statement handle. It then expects values to be provided for these parameters before the statement is executed. If you don’t bind a value to a parameter, SQLite will use NULL as the default when it executes the statement.

SQLite will use NULL as the default when it executes the statement. The advantage of parameter binding is that you can execute the same statement multiple times without having to recompile it.

You just reset the statement, bind a new set of values, and reexecute. This is where resetting rather than finalizing a statement comes in handy: it avoids the overhead of SQL compilation. By resetting a statement, you are reusing the compiled SQL code.

You completely avoid the tokenizing, parsing, and code generation overhead. Resetting a statement is implemented in the API by the sqlite3\_reset() function. The other advantage of parameters is that SQLite takes care of escaping the string values you bind to parameters.

For example, if you had a parameter value such as 'Kenny's Chicken', the parameter binding process will automatically convert it to 'Kenny''s Chicken'—escaping the single quote for you, helping you avoid syntax errors and possible SQL injection attacks

**Scraping to a Database Part 1**

**Scraping to a Database Part 2**