### Working With Python and Databases

Here are some notes from my other courses on this subject. Hope you find some of it useful!

### Connecting to a SQL Server Database

You can **download and installed Python modules** that allow you to connect to databases. One that connects to databases is called **PyPyODBC**. (<https://pypi.python.org/pypi/pypyodbc>)

#### PIP

To install modules, you use the Python package installer (pip). This is updated so can get the latest version of pip like this:

$ python3 -m pip install --upgrade pip

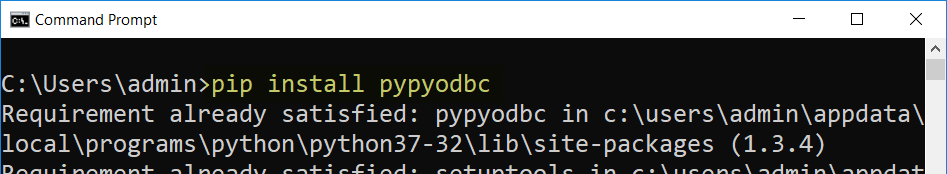
To use pip to install a package, you invoke it with this command:

python3 -m pip install the\_name\_of\_the\_package

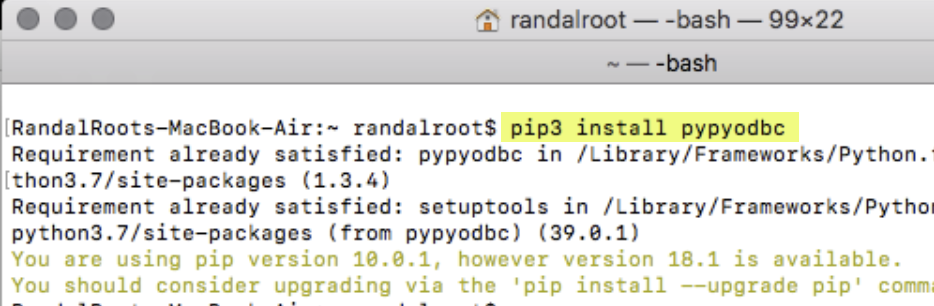
***NOTE: In Windows OS it is just Python, not python3***

#### Installing PyPyODBC

In a **Windows OS** you open can command window and run "**python -m pip install pypyodbc**" or "**python3 -m** **pip3** **install pypyodbc"** on **Mac OS**.



On **Mac OS**, you open a terminal window and run the command "**pip3 install pypyodbc**"



***Note: You can get more information here:***

* Windows -<https://pip.pypa.io/en/stable/user_guide/>
* Mac - <https://github.com/jiangwen365/pypyodbc/wiki/How-to-use-pypyodbc-on-MacOS-OSX>

#### Mac Issues

On Mac, things get a bit complicated! You need to perform a few additional installations to get things to work!

"This article explains how to install the Microsoft ODBC Driver for SQL Server on Linux and macOS, as well as the optional Command-Line Tools for SQL Server (bcp and sqlcmd) and the unixODBC Development Headers." (<https://docs.microsoft.com/en-us/sql/connect/odbc/linux-mac/installing-the-microsoft-odbc-driver-for-sql-server?view=sql-server-2017>, 2018)

##### HomeBrew (<https://brew.sh/>)

"Homebrew is a free and open-source software package management system that simplifies the installation of software on Apple's macOS operating system. Originally written by Max Howell, the package manager has gained popularity in the Ruby on Rails community and earned praise for its extensibility." ([Wikipedia](https://en.wikipedia.org/wiki/Homebrew_(package_management_software)), 2018)

To install on Mac OS, perform these commands in a terminal:

1. Install HomeBrew.

/usr/bin/ruby -e "$(curl -fsSL <https://raw.githubusercontent.com/Homebrew/install/master/install>)"

2. Update HomeBrew.

brew update

3.Access the MS SQL ODBC Drivers by tapping into its GitHub files.

brew tap microsoft/mssql-release https://github.com/Microsoft/homebrew-mssql-release

4. Use Brew to download and install Microsoft's ODBC drivers for SQL.

brew install microsoft/msodbcsql mssql-tools

5. Restart your Mac!

After that, you should be able to connect and run SQL commands!

### Using an ODBC driver

“Microsoft have written and distributed multiple ODBC drivers for SQL Server:

* {SQL Server} - released with SQL Server 2000
* {SQL Native Client} - released with SQL Server 2005 (also known as version 9.0)
* {SQL Server Native Client 10.0} - released with SQL Server 2008
* {SQL Server Native Client 11.0} - released with SQL Server 2012
* {ODBC Driver 11 for SQL Server} - supports SQL Server 2005 through 2014
* {ODBC Driver 13 for SQL Server} - supports SQL Server 2005 through 2016
* {ODBC Driver 13.1 for SQL Server} - supports SQL Server 2008 through 2016
* {ODBC Driver 17 for SQL Server} - supports SQL Server 2008 through 2017”

(https://github.com/mkleehammer/pyodbc/wiki/Connecting-to-SQL-Server-from-Windows)

### Connecting Python to SQL Server

Then **create a Python script** **that connects to the database**, start by testing the connection.

**import** pypyodbc

db\_driver=**'{ODBC Driver 13 for SQL Server}'**db\_host = **'is-root01.ischool.uw.edu'**db\_name = **'Northwind'**db\_user = **'Info330'**db\_password = **'sql'**connection\_string = **'Driver='** + db\_driver  
connection\_string += **';Server='** + db\_host  
connection\_string += **';Database='** + db\_name  
connection\_string += **';UID='** + db\_user  
connection\_string += **';PWD='** + db\_password + **';'**

objCon = pypyodbc.connect(connection\_string)  
print(**'It worked!'**)  
objCon.close

### Creating Report Objects

Once this is done, you **create reporting objects** in then database, like this reporting stored procedure:

If (Object\_ID('pSelSalesByCategory') != '')

Drop Proc pSelSalesByCategory;

Go

Create Proc pSelSalesByCategory

(@CategoryName nvarchar(100))

AS

Select Year(o.OrderDate) as OrderYear

,Sum(od.Quantity) as TotalQuantity

From Northwind.dbo.Categories as c

Join Northwind.dbo.Products as p

On c.CategoryID = p.CategoryID

Join Northwind.dbo.[Order Details] as od

On p.ProductID = od.ProductID

Join Northwind.dbo.Orders as o

On od.OrderID = o.OrderID

Where c.CategoryName = @CategoryName

Group By c.CategoryName, Year(o.OrderDate)

Order By CategoryName, OrderYear

Go

Grant Exec on pSelSalesByCategory to Public;

### Running SQL Code from Python

Next, let's **create a Python script** that connects to the database and executes the SQL code.

**import** pypyodbc

db\_driver=**'{ODBC Driver 13 for SQL Server}'**db\_host = **'is-root01.ischool.uw.edu'**db\_name = **'Northwind'**db\_user = **'Info330'**db\_password = **'sql'**connection\_string = **'Driver='** + db\_driver  
connection\_string += **';Server='** + db\_host  
connection\_string += **';Database='** + db\_name  
connection\_string += **';UID='** + db\_user  
connection\_string += **';PWD='** + db\_password + **';'**

objCon = pypyodbc.connect(connection\_string)  
  
objCursor = objCon.cursor()  
strCategory = **"Seafood"**objCursor.execute(**"Exec pSelSalesByCategory @CategoryName = "** + strCategory )  
print(**"Data for category: "** + strCategory)  
**for** row **in** objCursor:  
 print(str(row[0]) + **','** + str(row[1]))  
  
objCursor.close  
objCon.close

spring20b

## Other Programming Languages

There are **many** different **languages** and technologies used to create data-driven applications. You can **expect** them to follow the **same pattern** as seen in both the Python and C# examples:

* **Open** a **connection**
* **Issue** a **command**
* **Process** any **results**
* **Close** a **connection**

While the code may be a bit different in every language, if you **look for this pattern,** you should be able to figure out how the code works!

## SQLite

For our data layer**, we will create a new database using SQLite instead of MS SQL Server**. As you learn about SQLite database software, **note** **how** **your existing knowledge and experience can be applied to something new!**

SQLite is a simple SQL relational database used to implement **small, fast, self-contained, high-reliability, full-featured, SQL databases**. SQLite is **the most used database engine in the world**.

"SQLite is **built into all mobile phones** **and** most **computers** and comes bundled inside countless other applications that people use every day." (<https://www.sqlite.org/index.html>, 2019)

### Installing SQLite

"SQLite is famous for its great feature zero-configuration, which means **no complex setup or administration is needed**. This chapter will take you through the process of setting up SQLite on Windows, Linux and Mac OS X." <https://www.tutorialspoint.com/sqlite/sqlite_installation.htm>

### Download SQLite

The first step in the installation is to download the software. You can get it at: <https://www.sqlite.org/download.html>

Be careful to download the correct version for your computer's OS.

***NOTE:*** *"Nowadays, Most of the* ***Mac OS X*** *(edit: and* ***Linux****) distributions, if not all, are being* ***shipped with Sqlite****. So* ***before installing*** *Sqlite manually, it's not a bad idea to* ***check if the SQLite is installed******on your Mac OS X*** *system or not."*

…

**So, open your terminal and type sqlite3** in it. If you the following result on the terminal, then SQLite is already present on your **Mac OS X** operating system.

$sqlite3

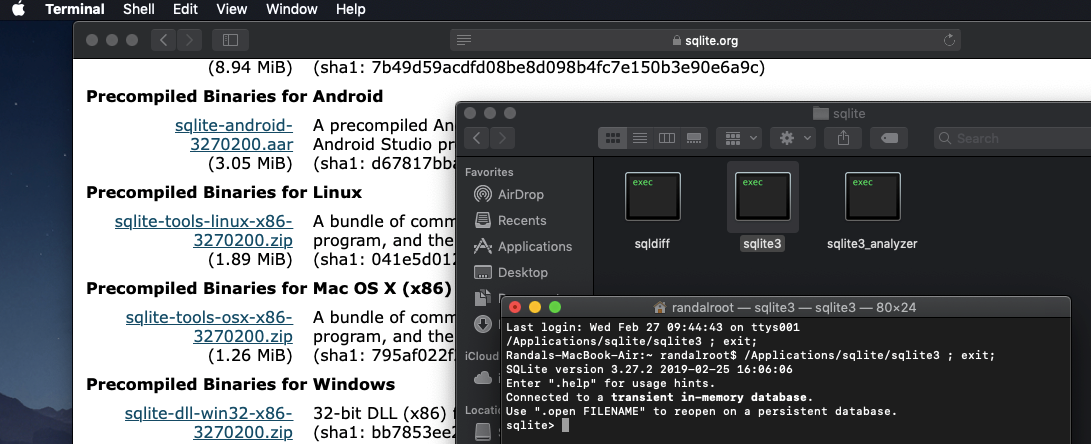
SQLite version 3.13.0 2016-05-18 10:57:30

…

### Installing on Mac

(<http://www.codebind.com/sqlite/how-to-install-sqlite-on/>, 2019)

If you do not find SQLite on your Mac, just download the latest version then unzip it in you Applications folder:



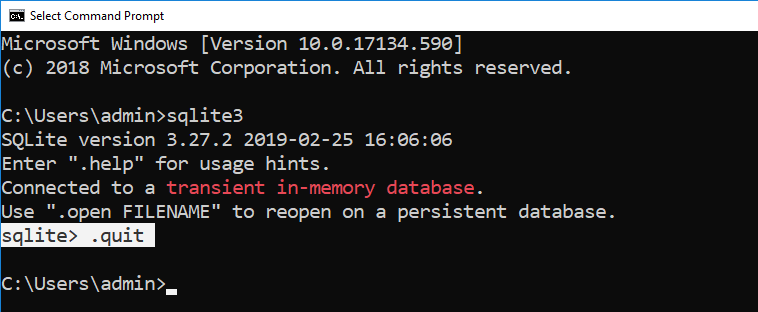
Once it is installed, you can **use it by navigating to its folder** (or modifying the OS's path) and **launching** the applications using "**SQLite3.exe**."

Using "**SQLite3.exe**." opens a **SQLite command** prompt where you can use standard SQL commands to create, store, and manipulate data.

**Tip**: You can quit the command prompt using the "**.quit**" command.

### Installing on Windows

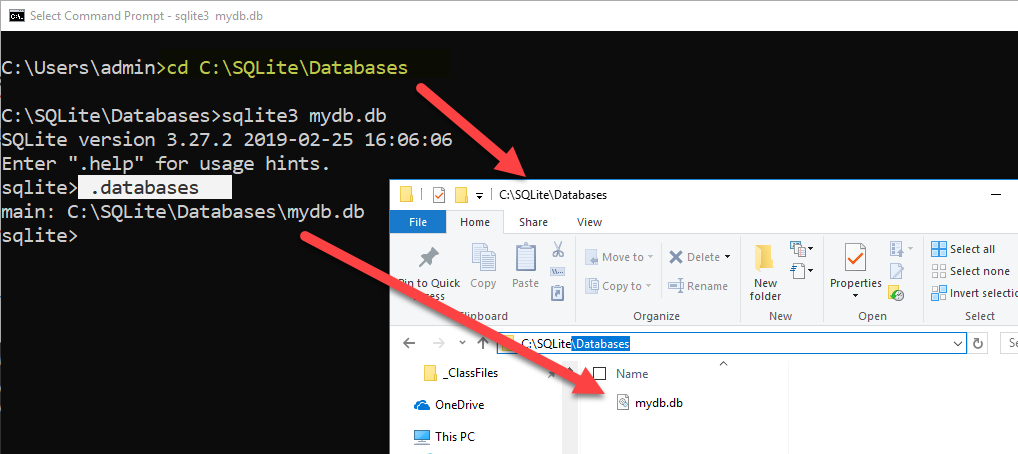
Installing on Windows is pretty much the same, but you must place it in a different folder of your choosing. Make sure to put the unzipped files somewhere that is easy to navigate to as shown in this example:



## SQLite Databases

**Once you have SQLite installed, you need to make a database to work with it. To create a database** in SQLite, you open it using this command, "**sqlite3.exe <Name of database>.db**!

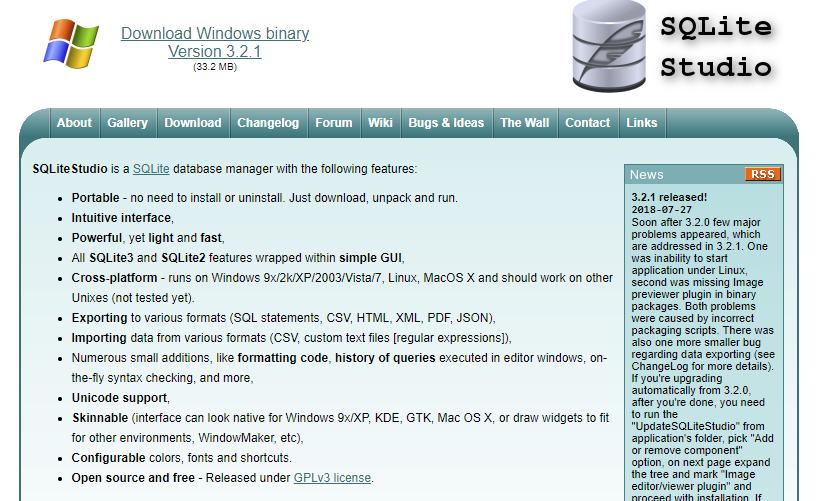
**NOTE:** At first, this will only **create an in-memory version of the database, but not an actual file** on your computer. You can **list the currently open databases** using the "**.databases**" command and doing so **will materialize the database onto your computer**.



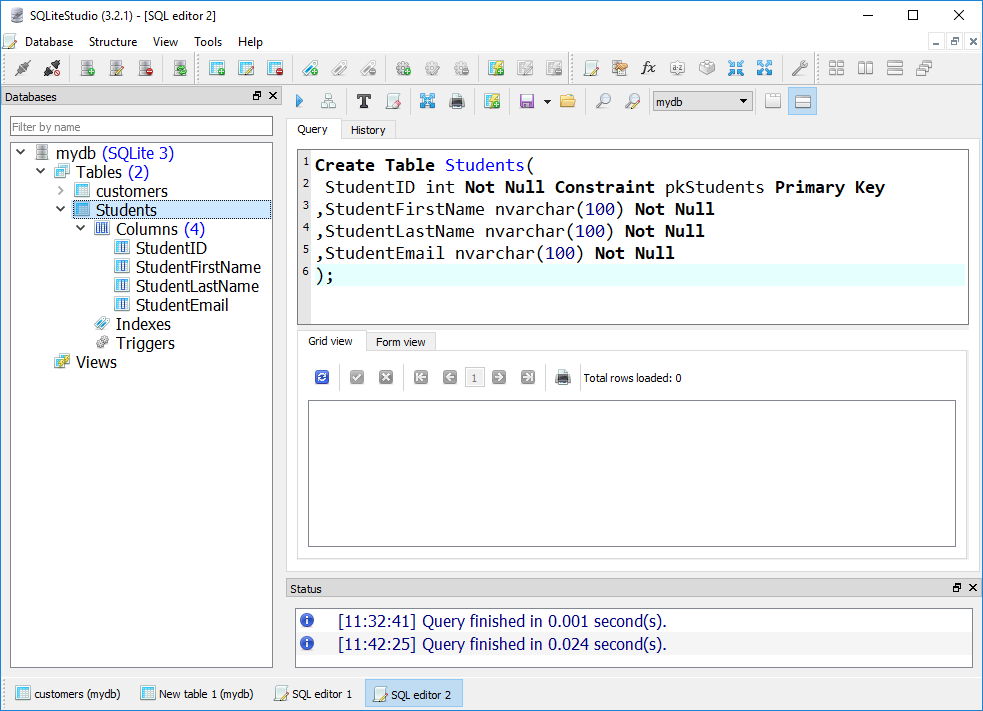
#### The SQLite Studio GUI Editor

If you want **a UI for working with SQLite**, download the free **SQLiteStudio** for **Mac, Linux, and Windows** ( <https://sqlitestudio.pl>).

**Note:** For Mac users read this article which includes changing a security option: <https://www.dev2qa.com/how-to-install-sqlite3-on-mac/>



Once you have **downloaded and installed SQLite Studio**, you need to launch and see how it works! Since you have some experience with using a SQL editor, you should be able to figure out quite a bit of this new software's functionality on your own.



## SQLite Tables

Creating a table will materialize the new database file if it has not already done so. **To create a table**, you use the standard SQL code like this:

**create** **table** customers**(**CustomerID int primary key, CustomerName char(100)**);**

### Column datatypes

Different from other database systems, **SQLite uses a *dynamic type system***. For example, the value stored in a column determines its data type, not the column’s data type.

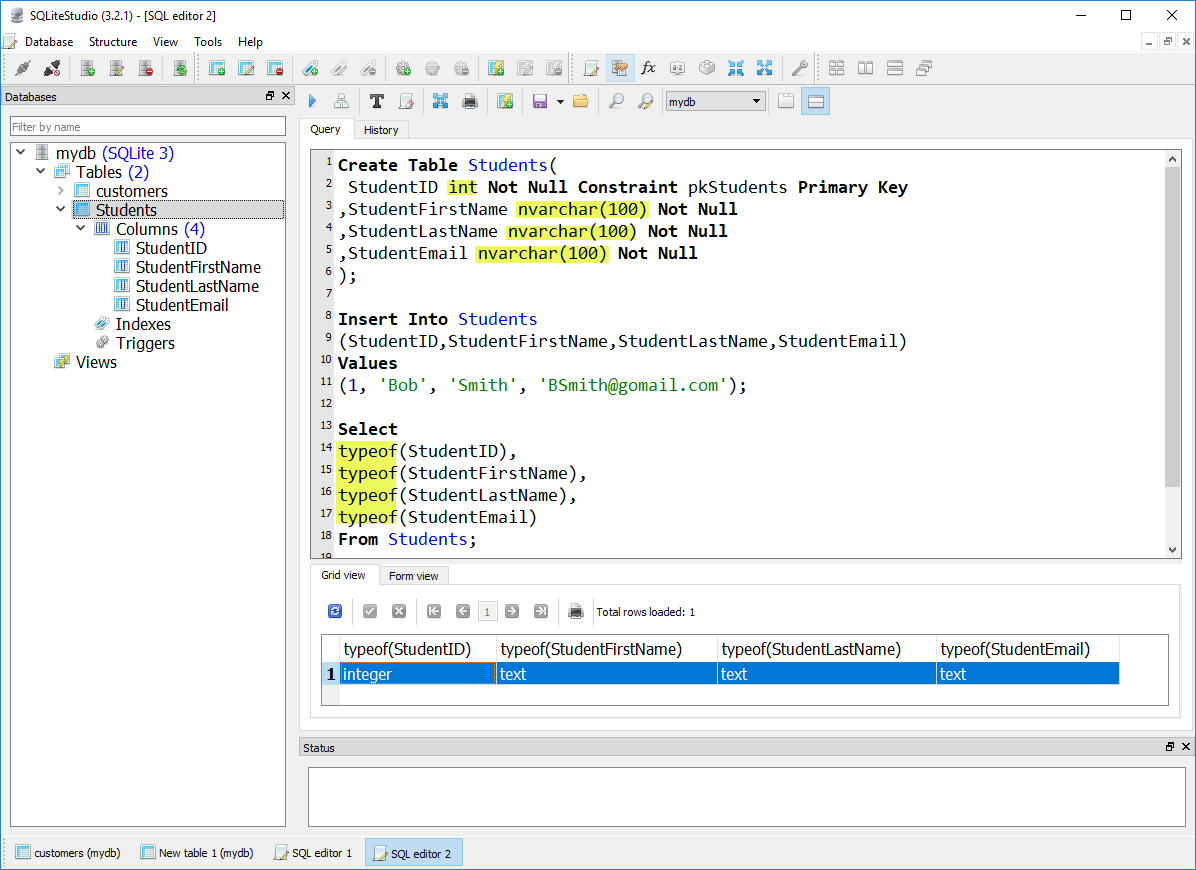
Besides, you don’t have to declare a specific data type for a column when you create a table. In case you declare a column with the integer data type, you can store any kind of data types such as text and BLOB, SQLite will not complain about this.

That said, there are some **basic categories of data** that are used. These are **known as "Storage Classes."**

"SQLite provides **five primitive data types** which referred to as storage classes.

The concept of **storage classes** describes the format SQLite uses to store data on disk. A storage class is **more general than a data type** e.g., INTEGER storage class includes 6 different types of integers. In most cases, you can use storage classes and data type interchangeably." (<http://www.sqlitetutorial.net/sqlite-data-types/>, 2019)

**Like Python, the data type of** **data is chosen based on its value**. So, if you insert data into the table, a data class will be selected for the data **automatically**. It is up to you to perform validation!



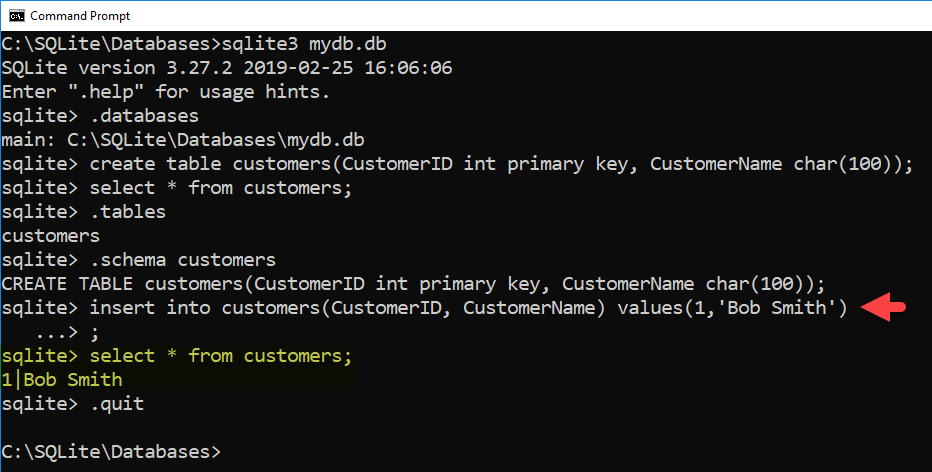
**Notes:**

* SQLite **automatically begins and commits the transaction**. You are not required to use "begin," "commit," or "rollback" transaction statements.
* Since SQLite is serverless, made to run on a single device, it **doesn't have stored procedures**! Validation logic must be placed either in triggers or in your application code!

### Inserting Data

If the table is empty, you will not see any results from your select statement. To add data to a table, you use a SQL insert command, like this:

**Note**: I am showing the command shell version this time for variety!



**Note**: If you forget the **semi-colon,** the code **will not run until one is added**!

## PySQLite

Once we have the database software installed and tested, **we need to set up and test the application software**. Python includes an **easy way to connect to your SQLite database from Python** **using the PySQLite** **module**.

"pysqlite is an interface to the SQLite 3.x embedded relational database engine." ( <https://pypi.org/project/pysqlite/> ,2019)

**Note**: PySQLite is part of the Python standard library and **should be installed** with Python. **If not**, it **can be installed or updated** using pip ( *pip install pysqlite* ).

To test the installation, you can create the following code:

import sqlite3

con = sqlite3.connect('C:/sqlite/databases/Enrollments.db')

Important: Make sure to change the path to the location on your computer! Mac users can use this page to find the path. <https://osxdaily.com/2015/11/05/copy-file-path-name-text-mac-os-x-finder/>

## Connecting to SQLite with PySQLite

With SQLite installed, you can **create new or connect to existing SQLite databases by using PySQLite in a Python Script. You do so using the connection() method as shown here:**

**import** sqlite3 # this imports code from the SQLite module of PySQLite!  
  
# Functions -----------------------------------------------------  
**def** create\_connection(db\_file):  
 **try**:  
 con = sqlite3.connect(db\_file) # This opens OR creates the database  
 print(**'Connected! - SQLite Version is: '**, sqlite3.version)  
 **except** Exception **as** e:  
 print(e.\_\_str\_\_())  
 **return** con  
  
# Main body of the script------------------------------------------  
db\_con = create\_connection(**'C:/DataFiles/test.db'**)   
db\_con.close() # Always close the connection when your done

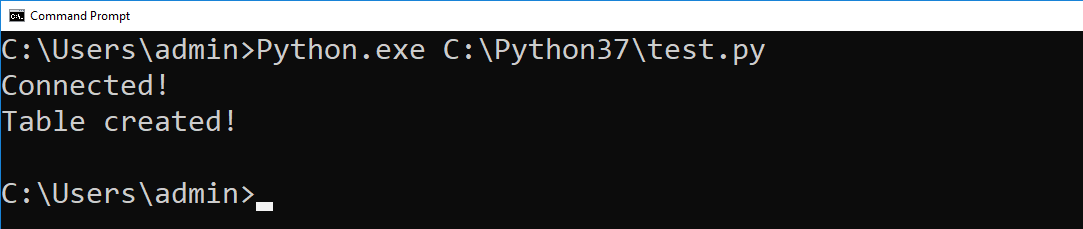
**Notes**:

* If the database file does not exist, **it will create it**!
* You can **specify a path** to the database file like **'C:/DataFiles/test.db**' on Windows and **'~/Desktop/Datafiles/test.db'** on Mac.
* If you **do not specify a path**, the database will be **created in the same folder you are running Python.exe from**!

## Executing SQL Code from Python

Now that we have created a database, we need to use it! You use the **execute**() function of a **sqlite3.cursor object** to submit SQL code to the database. Here is an example:

**import** sqlite3  
  
*# Functions ----------------------------------------------------***def create\_connection**(db\_file):  
 **try**:  
 con = sqlite3.connect(db\_file) *# This opens OR creates the database* **except** Exception **as** e:  
 **raise** e  
 **return** con  
  
  
**def create\_demo\_table**(con):  
 **try**:  
 csr = con.cursor() *# A cursor object allows you to submit commands* csr.execute(**"Create Table Demo (ID [integer], Name [text]);"**) csr.close() *# Always close the cursor when your done* **except** Exception **as** e:  
 **raise** e  
  
  
*# Main body of the script ------------------------------------*db\_con = **None  
  
try**: *# Connecting* db\_con = **create\_connection**(**'test.db'**)  
 print(**"Connected!"**)  
**except** Exception **as** e:  
 print(e)  
  
**try**: *# Creating* **create\_demo\_table**(db\_con)  
 print(**"Table created!"**)  
**except** Exception **as** e:  
 print(e)

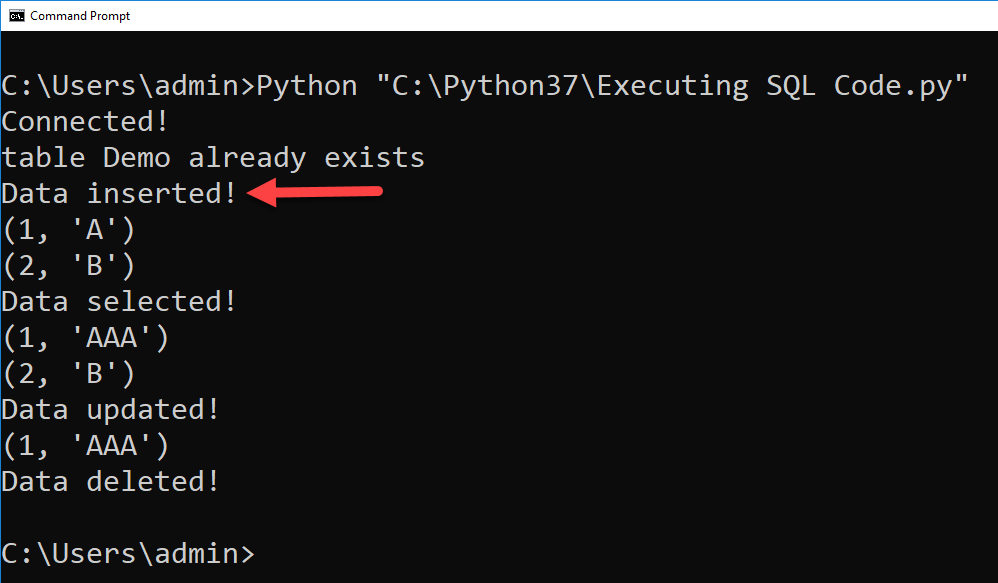


### Executing Transaction Statements

Unlike using SQLite directly, when **using PySQLite any transaction statement** (insert, update, or delete) **must include a commit** statement.

**def insert\_demo\_data**(con):  
 **try**:  
 csr = con.cursor() *# A cursor object allows you to submit commands* csr.execute(**"Insert Into Demo (ID, Name) values (1, 'A'), (2, 'B');"**)

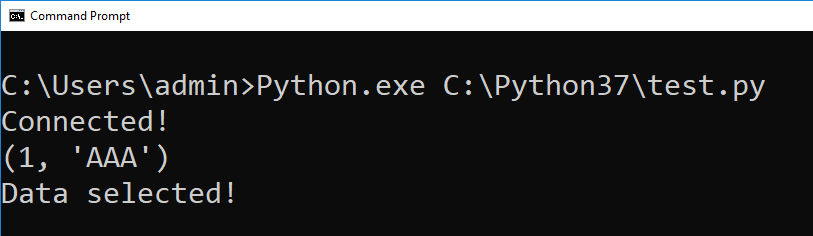
csr.execute(**"commit;"**) *# You need to add this when using PySQLite!* csr.close() *# Always close the cursor when your done* **except** Exception **as** e:  
 **raise** e  
  
**def update\_demo\_data**(con):  
 **try**:  
 csr = con.cursor() *# A cursor object allows you to submit commands* csr.execute(**"Update Demo Set Name = 'AAA' Where ID = 1;"**) *# Single quotes* csr.execute(**"commit;"**) *# You need to add this when using PySQLite!* csr.close() *# Always close the cursor when your done* **except** Exception **as** e:  
 **raise** e  
  
**def delete\_demo\_data**(con):  
 **try**:  
 csr = con.cursor() *# A cursor object allows you to submit commands* csr.execute(**"Delete From Demo Where ID = 2;"**) *# Single quotes for strings!* csr.execute(**"commit;"**) *# You need to add this when using PySQLite!* csr.close() *# Always close the cursor when your done* **except** Exception **as** e:  
 **raise** e



### Executing Select Statements

If the SQL statement is a **SELECT**, you can **use** the **fetchall**() method of a sqlite3.cursor object to retrieve the results.

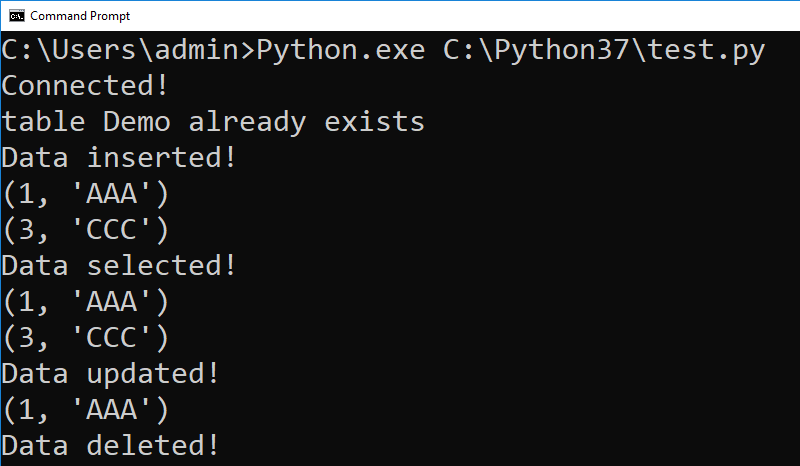
**def** select\_demo\_data(con):  
 **try**:  
 csr = con.cursor() *# A cursor object allows you to submit commands* csr.execute(**"Select ID, Name From Demo;"**)  
 rows = csr.fetchall() *# fetchall puts all rows from the result into a list* csr.close() *# Always close the cursor when your done* **return** rows  
 **except** Exception **as** e:  
 **raise** e  
  
*# Main body of the script ------------------------------------*db\_con = **None  
  
try**: *# Connecting* db\_con = create\_connection(**'test.db'**)  
 print(**"Connected!"**)  
**except** Exception **as** e:  
 print(e)  
  
**try**: *# Selecting* rows = select\_demo\_data(db\_con)  
 **for** row **in** rows:  
 print(row)  
 print(**"Data selected!"**)  
**except** Exception **as** e:  
 print(e)



Using Parameterized Commands

You can use **parameters to create dynamic SQL statements**. PySQLite, makes this easy by allowing you to **pass in a list of arguments** to configure a SQL string with **question marks for parameters**.

**def** insert\_demo\_data(con, ID, Name):  
 **try**:  
 csr = con.cursor()  
 csr.execute(**"INSERT INTO Demo (ID, Name) values (?,?);"**, [ID, Name])  
 csr.execute(**"commit;"**)  
 csr.close()  
 **except** Exception **as** e:  
 **raise** e  
  
**def** update\_demo\_data(con, ID, Name):  
 **try**:  
 csr = con.cursor()ds  
 csr.execute(**"Update Demo Set Name = ? Where ID = ?;"**, [ID, Name])  
 csr.execute(**"commit;"**)  
 csr.close()  
 **except** Exception **as** e:  
 **raise** e  
  
**def** delete\_demo\_data(con, ID):  
 **try**:  
 csr = con.cursor()  
 csr.execute(**"Delete From Demo Where ID = ?;"**, [ID])  
 csr.execute(**"commit;"**)  
 csr.close()  
 **except** Exception **as** e:  
 **raise** e



Here is the **completed version of the code**:

**import** sqlite3  
  
*# Functions ----------------------------------------------------***def** create\_connection(db\_file):  
 **try**:  
 con = sqlite3.connect(db\_file) *# This opens OR creates the database* **except** Exception **as** e:  
 **raise** e  
 **return** con  
  
**def** create\_demo\_table(con):  
 **try**:  
 csr = con.cursor() *# A cursor object allows you to submit commands* csr.execute(**"Create Table Demo (ID [integer] Primary Key, Name [text]);"**)  
 csr.close() *# Always close the cursor when your done* **except** Exception **as** e:  
 **raise** e  
  
**def** insert\_demo\_data(con, ID, Name):  
 **try**:  
 csr = con.cursor()  
 csr.execute(**"INSERT INTO Demo (ID, Name) values (?,?);"**, [ID, Name])  
 csr.execute(**"commit;"**)  
 csr.close()  
 **except** Exception **as** e:  
 **raise** e  
  
**def** update\_demo\_data(con, ID, Name):  
 **try**:  
 csr = con.cursor()  
 csr.execute(**"Update Demo Set Name = ? Where ID = ?;"**, [Name, ID])  
 csr.execute(**"commit;"**)  
 csr.close()  
 **except** Exception **as** e:  
 **raise** e  
  
**def** delete\_demo\_data(con, ID):  
 **try**:  
 csr = con.cursor()  
 csr.execute(**"Delete From Demo Where ID = ?;"**, [ID])  
 csr.execute(**"commit;"**)  
 csr.close()  
 **except** Exception **as** e:  
 **raise** e  
  
**def** select\_demo\_data(con):  
 **try**:  
 csr = con.cursor() *# A cursor object allows you to submit commands* csr.execute(**"Select ID, Name From Demo;"**)  
 rows = csr.fetchall() *# fetchall puts all rows from the result into a list* csr.close() *# Always close the cursor when your done* **return** rows  
 **except** Exception **as** e:  
 **raise** e  
  
**def** print\_selected\_data():  
 **try**: *# Selecting* rows = select\_demo\_data(db\_con)  
 **for** row **in** rows:  
 print(row)  
 print(**"Data selected!"**)  
 **except** Exception **as** e:  
 print(e)  
  
*# Main body of the script ------------------------------------*db\_con = **None  
  
try**: *# Connecting* db\_con = create\_connection(**'test.db'**)  
 print(**"Connected!"**)  
**except** Exception **as** e:  
 print(e)  
  
**try**: *# Creating* create\_demo\_table(db\_con)  
 print(**"Table created!"**)  
**except** Exception **as** e:  
 print(e)  
  
**try**: *# Creating* insert\_demo\_data(db\_con, 1, **'test insert'**)  
 print(**"Insert Committed"**)  
 print\_selected\_data() *# Show the change***except** Exception **as** e:  
 print(e)  
  
**try**: *# Creating* update\_demo\_data(db\_con, 1, **'test update'**)  
 print(**"Update Committed"**)  
 print\_selected\_data() *# Show the change***except** Exception **as** e:  
 print(e)  
  
**try**: *# Creating* delete\_demo\_data(db\_con, 1)  
 print(**"Delete Committed"**)  
 print\_selected\_data() *# Show the change***except** Exception **as** e:  
 print(e)  
  
db\_con.close() *# Always close the connection when your done*

Here is another example:

Create a **new Database called Enrollments**.

**Important:** Place the database file in a folder called "DataFiles"!

Create a **new table called Students** using the following code:

CREATE TABLE Students (

StudentID INT NOT NULL CONSTRAINT pkStudents PRIMARY KEY,

StudentFirstName NVARCHAR (100) NOT NULL,

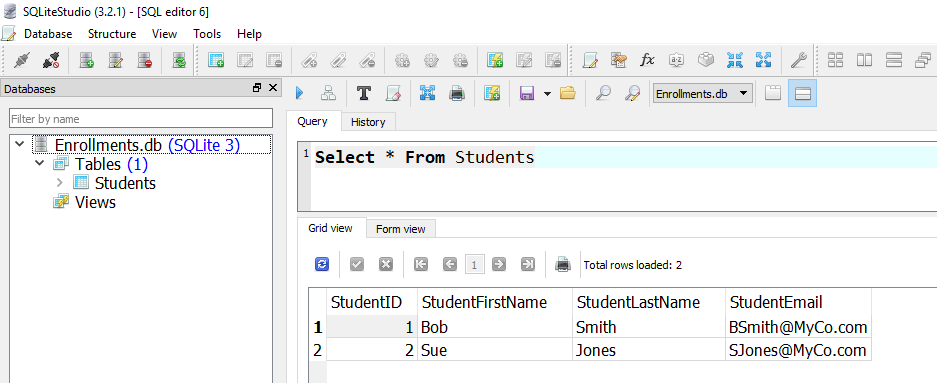
StudentLastName NVARCHAR (100) NOT NULL,

StudentEmail NVARCHAR (100) NOT NULL);

Add some test data to the table, then select that data to prove that the insert worked. Then create the following Python script.

**import** sqlite3  
  
*# Functions ----------------------------------------------------***def** create\_connection(db\_file):  
 **try**:  
 con = sqlite3.connect(db\_file) *# This opens OR creates the database* **except** Exception **as** e:  
 **raise** e  
 **return** con  
  
**def** insert\_students\_data(con, ID, FName, LName, Email):  
 **try**:  
 csr = con.cursor()  
 csr.execute(**"INSERT INTO Students values (?,?,?,?);"**, [ID, FName, LName, Email])  
 csr.execute(**"commit;"**)  
 csr.close()  
 **except** Exception **as** e:  
 **raise** e  
  
  
*# Main body of the script ------------------------------------*db\_con = **None  
  
try**: *# Connecting* db\_con = create\_connection(**'C:/DataFiles/Enrollments.db'**)  
 print(**"Connected!"**)  
**except** Exception **as** e:  
 print(e)  
  
**try**: *# Inserting* insert\_students\_data(db\_con, 2, **"Sue"**, **"Jones"**, **"SJones@MyCo.com"**)  
 print(**"Data inserted!"**)  
**except** Exception **as** e:  
 print(e)

**Verify** that the data was inserted **using SQLiteStudio**.



Now add code to select data from the Students table.

**def** select\_students\_data(con):  
 **try**:  
 csr = con.cursor()  
 csr.execute(**"SELECT StudentID, StudentFirstName, StudentLastName, StudentEmail FROM Students;"**)  
 rows = csr.fetchall()  
 csr.close()  
 **return** rows  
 **except** Exception **as** e:  
 **raise** e  
  
*# Main body of the script ------------------------------------*db\_con = **None  
  
try**: *# Connecting* db\_con = create\_connection(**'C:/DataFiles/Enrollments.db'**)  
 print(**"Connected!"**)  
**except** Exception **as** e:  
 print(e)  
  
**try**: *# Inserting* insert\_students\_data(db\_con, 2, **"Sue"**, **"Jones"**, **"SJones@MyCo.com"**)  
 print(**"Data inserted!"**)  
**except** Exception **as** e:  
 print(e)  
  
**try**: *# Selecting* rows = select\_students\_data(db\_con)  
 **for** row **in** rows:  
 print(row)  
 print(**"Data selected!"**)  
**except** Exception **as** e:  
 print(e)

