In order to view the files in sqlite database:

1. Drag and drop the database in this link:

<https://inloop.github.io/sqlite-viewer/>

1. Open the anaconda prompt -> go the folder where we have our sqlite database -> write in the command: sqlite3 hawaii.sqlite -> we get 'sqlite>’ on the cmd -> .tables – to see the tables in the db -> write normal queries ending with a ;
2. When we write the query ‘select \* from eausrement;” -> we get the output but we do not get the colum names. In order to also view the column names with the data, write the command: ‘**.headers on**’. This will turn the headers on and we will be able to see the column names.
3. As of now our data is separated by pipes. We can change this by changing the mode : ‘.mode csv’ -> will show us the data in a comma separated format. If we do ‘.mode column’ then we see the data in neat columns.

Sqlite is a Relational Database. Lives on a disk.

It is:

1. Light
2. Easy
3. Powerful
4. Most commonly used

Create a SQLITE database in VSCODE:

import sqlite3

# define connection and cursor. We need these to connect to our database

connection = sqlite3.connect('store\_transactions1.sqlite')

#Create acursor to interact with our database through sql commands. This allows to create or modify the tables within our database.

cursor = connection.cursor()

#Create stores table

command1 = """CREATE TABLE IF NOT EXISTS

STORES(STORES\_ID INTEGER PRIMARY KEY, LOCATION TEXT)"""

cursor.execute(command1)

#Create purchases table

command2 = """create table if not exists

 purchases(purchase\_id integer primary key, store\_id integer, total\_cost float,

 foreign key (store\_id) references stores(store\_id))"""

cursor.execute(command2)

cursor.execute("insert into stores values(01, 'austin')")

cursor.execute("insert into stores values(02, 'dallas')")

cursor.execute("insert into stores values(03, 'houston')")

cursor.execute("insert into stores values(04, 'round rock')")

cursor.execute("insert into stores values(05, 'plano')")

cursor.execute("insert into purchases values(10,01, 100)")

cursor.execute("insert into purchases values(20,02, 200)")

cursor.execute("insert into purchases values(30,03, 300)")

cursor.execute("insert into purchases values(40,04, 400)")

cursor.execute("insert into purchases values(50,05, 500)")

connection.commit()

connection.close()

# results = cursor.fetchall()

# print(results)

We can create .db and .sqlite databases from the above code. When I try running both , they worked in the same fashion.

**SQLAMCHEMY**

SQLAlchemy is a Python library that works across a variety of SQL dialects. Write query once , run it everywhere. Pandas integrates with SQLAlchemy

# Create an engine to connect with the database

engine = create\_engine("sqlite:///hawaii.sqlite")

|  |  |  |
| --- | --- | --- |
| we can **directly query the database** if we are aware of the tables: | We can **query** the database using **by creating a session**: | **Querying using Pandas** |
| data = engine.execute(“select \* from measurement”) 🡨 The database ‘hawaii’ contains 2 tables- measurement, station | # reflect an existing database into a new model  Base = automap\_base()  # reflect the tables  Base.prepare(engine, reflect=True)  # We can view all of the classes that automap found  Base.classes.keys()  # Save references to each table  Measurement = Base.classes.measurement  Station = Base.classes.station  results = session.query(Measurement.date, Measurement.prcp) | conn = engine.connect()  data = pd.read\_sql("select \* from measurement limit 10",conn)  data |
| print(data.all()) 🡨-- prints the data  Result is a List | Print(results.all())  Result is a List |  |