Name: Kelly Joseph Calvadores
Course and Section: CPE 019 - CPE32S3
Date of Submission: January 31, 2024
Instructor: Engr. Roman M. Richard

Working with Python and SQLite

Objectives:
Use the sqlite3 module to interact with a SQL database.
Access data stored in SQLite using Python.
Describe the difference in interacting with data stored as a CSV file versus in SQLite.
Describe the benefits of accessing data using a database compared to a CSV file.

Required Resources

1 PC with Internet access
Jupyter notebook

SQL refresh

Very brief introduction to relational databases (temporary): http://searchsqlserver.techtarget.com/definition/relational-database More videos on relational databases: https://www.youtube.com/watch?v=jyju2P-7hPA&list=PLAwxTw4SYaPm4R6j_wzVOCV9fJaiQDYx4 Introduction to SQL: http://www.w3schools.com/sql/sql_intro.asp Working with SQLite via the command-line: https://www.sqlite.org/cli.html

Step 1: Create a SQL connection to our SQLite database Creating a new SQLite database is as simple as creating a connection using the sqlite3 module in the Python standard library. To establish a connection all you need to do is pass a file path to the connect(...) method in the sqlite3 module, and if the database represented by the file does not exists one will be created at that path.

```
import sqlite3
con = sqlite3.connect("sqlite.db")
```

```
import os

#Create default path to create and connect database
DefaultPath = os.path.join("sqlite3.db")

#Function for connection
def dbConnect(dbPath = DefaultPath):
  con = sqlite3.connect(dbPath)
  return con
```

Step 2: Create a table on the SQLite database
The code below creates a table on the sqlite.db database. The cursor() command is needed to make a cursor object to interact with the created database. The cursor is then ready to perform all kinds of operations with .execute(). The execute() command performs a query that creates a table using the parameters as shown. The commit() command

In order to create database tables you need to have an idea of the structure of the data you are interested in storing. There are many design considerations that go into defining the tables of a relational database. To aid in the discussion of SQLite database programming with Python, we will be working off the premise that a database needs to be created for a fictitious book store that has the below data already collected on book sales.

Customer	Date	Product	Price
Allan Turing	2/22/1944	Introduction to Combinatorics	7.99
Donald Knuth	7/3/1967	A Guide to Writing Short Stories	17.99
Donald Knuth	7/3/1967	Data Structures and Algorithms	11.99
Edgar Codd	1/12/1969	Advanced Set Theory	16.99

Upon inspecting this data, it is evident that it contains information about customers, products, and orders. A common pattern in database design for transactional systems of this type are to break the orders into two additional tables, orders and line items (sometimes referred to as order details) to achieve greater normalization.

Enter the SQL for creating the customers and products tables follows:

```
con = dbConnect()
cur = con.cursor()
customers_sql = """CREATE TABLE customers(id integer PRIMARY KEY, first_name text NOT NULL, last_name text NOT NULL)"""
cur.execute(customers_sql)
product_sql = """CREATE TABLE products(id integer PRIMARY KEY, name text NOT NULL, price real NOT NULL)"""
cur.execute(product_sql)
```

```
<sqlite3.Cursor at 0x7c3572f670c0>
```

```
#To see all tables inside the database that is currently using
cur.execute("SELECT name FROM sqlite_master WHERE type = 'table'")
print(cur.fetchall())
```

```
[('customers',), ('products',)]
```

```
#To see the context of the table
cur.execute("SELECT sql FROM sqlite_master WHERE type = 'table'AND name = 'products'")
print(cur.fetchall()[0])
```

```
('CREATE TABLE customers(id integer PRIMARY KEY, first_name text NOT NULL, last_name text NOT NULL)',)
```

Created a table called "lineitems" where listed all items that will be purchased This table is connected to 2 tables called "products" and "orders"

```
orders_sql = """
... CREATE TABLE orders (
...     id integer PRIMARY KEY,
...     date text NOT NULL,
...     customer_id integer,
...     FOREIGN KEY (customer_id) REFERENCES customers (id))"""
cur.execute(orders_sql)
```

Created a table called "lineitems" where listed all items that will be purchased This table is connected to 2 tables called "products" and "orders"

```
lineitems_sql = """
... CREATE TABLE lineitems (
...     id integer PRIMARY KEY,
...     quantity integer NOT NULL,
...     total real NOT NULL,
...     product_id integer,
...     order_id integer,
...     FOREIGN KEY (product_id) REFERENCES products (id),
...     FOREIGN KEY (order_id) REFERENCES orders (id))"""
cur.execute(lineitems_sql)
```

Step 3: Loading the Data

In this section we will use INSERT to our sample data into the tables just created. A natural starting place would be to populate the products table first because without products we cannot have a sale and thus would not have the foreign keys to relate to the line items and orders. Looking at the sample data, we see that there are four products:

1. Introduction to Combinatorics - 7.99
2. A Guide to Writing Short Stories -17.99
3. Data Structures and Algorithms - 11.99
4. Advanced Set Theory - 16.99

The workflow for executing INSERT statements is simply:

1. Connect to the database
2. Create a cursor object
3. Write a parameterized insert SQL statement and store as a variable
4. Call the execute method on the cursor object passing it the sql variable and the values, as a tuple, to be inserted into the table

Given this general outline let us write some more code.

```
#con = db_connect()
#cur = con.cursor()
product_sql = "INSERT INTO products (name, price) VALUES (?, ?)"
cur.execute(product_sql, ('Introduction to Combinatorics', 7.99))
cur.execute(product_sql, ('A Guide to Writing Short Stories', 17.99))
cur.execute(product_sql, ('Data Structures and Algorithms', 11.99))
cur.execute(product_sql, ('Advanced Set Theory', 16.99))
con.commit()
```

```
cur.execute("SELECT id, name, price FROM products")
formatted_result = [f"{id:<5}{name:<35}{price:>5}" for id, name, price in cur.fetchall()]
id, product, price = "Id", "Product", "Price"
print('\n'.join([f"{id:<5}{product:<35}{price:>5}"] + formatted_result))
```

```
Id   Product                            Price
1    Introduction to Combinatorics       7.99
2    A Guide to Writing Short Stories   17.99
3    Data Structures and Algorithms     11.99
4    Advanced Set Theory                16.99
```

```
customer_sql = "INSERT INTO customers (first_name, last_name) VALUES (?, ?)"
cur.execute(customer_sql, ('Alan', 'Turing'))
customer_id = cur.lastrowid
print(customer_id)
con.commit()
```

```
1
```

Task 1: Insert 3 more records on the customers table

Insert the following records:

1. Donald Knuth
2. Edgar Codd
3. Martin Forest

```
customer_sql = "INSERT INTO customers (first_name, last_name) VALUES (?, ?)"
cur.execute(customer_sql, ('Donald', 'Knuth'))
customer_id = cur.lastrowid
print(customer_id)
con.commit()
```

```
2
```

```
customer_sql = "INSERT INTO customers (first_name, last_name) VALUES (?, ?)"
cur.execute(customer_sql, ('Edgar', 'Codd'))
customer_id = cur.lastrowid
print(customer_id)
con.commit()
```

```
3
```

```
customer_sql = "INSERT INTO customers (first_name, last_name) VALUES (?, ?)"
cur.execute(customer_sql, ('Martin', 'Forest'))
customer_id = cur.lastrowid
print(customer_id)
con.commit()
```

```
4
```

```
order_sql = "INSERT INTO orders (date, customer_id) VALUES (?, ?)"
date = "1944-02-22" # ISO formatted date
cur.execute(order_sql, (date, customer_id))
order_id = cur.lastrowid
print(order_id)
con.commit()
```

```
1
```

Task 2: Insert 3 more records on the orders table

Insert the following records:

1. for Donald Knuth, date is 7/3/1967
2. Edgar Codd, date is 1/12/1969
3. Martin Forest, date is 1/15/2021

```
order_sql = "INSERT INTO orders (date, customer_id) VALUES (?, ?)"
date = "1967-07-03"
cur.execute(order_sql, (date, customer_id))
order_id = cur.lastrowid
print(order_id)
con.commit()
```

```
2
```

```
order_sql = "INSERT INTO orders (date, customer_id) VALUES (?, ?)"
date = "1969-01-12"
cur.execute(order_sql, (date, customer_id))
order_id = cur.lastrowid
print(order_id)
con.commit()
```

```
3
```

```
order_sql = "INSERT INTO orders (date, customer_id) VALUES (?, ?)"
date = "2021-01-15" # ISO formatted date
cur.execute(order_sql, (date, customer_id))
order_id = cur.lastrowid
print(order_id)
con.commit()
```

```
4
```

```
con.rollback()
```

```
li_sql = """INSERT INTO lineitems
...       (order_id, product_id, quantity, total)
...     VALUES (?, ?, ?, ?)"""
product_id = 1
cur.execute(li_sql, (order_id, 1, 1, 7.99))
con.commit()
```

Task 3: Insert 3 more records on the lineitems

Insert the following records:

1. for Donald Knuth, insert (order_id, 2, 2, 17.99)
2. Edgar Codd, insert (order_id, 3, 3, 11.99)
3. Martin Forest, insert (order_id, 4, 4, 10.99)

```
li_sql = """INSERT INTO lineitems
...       (order_id, product_id, quantity, total)
...     VALUES (?, ?, ?, ?)"""
product_id = 2
cur.execute(li_sql, (order_id, 2, 2, 17.99))
con.commit()
```

```
li_sql = """INSERT INTO lineitems
...       (order_id, product_id, quantity, total)
...     VALUES (?, ?, ?, ?)"""
product_id = 3
cur.execute(li_sql, (order_id, 3, 3, 11.99))
con.commit()
```

```
li_sql = """INSERT INTO lineitems
...       (order_id, product_id, quantity, total)
...     VALUES (?, ?, ?, ?)"""
product_id = 4
cur.execute(li_sql, (order_id, 4, 4, 10.99))
con.commit()
```

Step 3: Querying the Database

Generally the most common action performed on a database is a retrieval of some of the data stored in it via a SELECT statement. For this section, we will be demonstrating how to use the sqlite3 interface to perform simple SELECT queries.

To perform a basic multirow query of the customers table you pass a SELECT statement to the execute(...) method of the cursor object. After this you can iterate over the results of the query by calling the fetchall() method of the same cursor object.

```
cur.execute("SELECT * FROM customers")
results = cur.fetchall()
for row in results:
  print(row)
```

```
    (1, 'Alan', 'Turing')
    (2, 'Donald', 'Knuth')
    (3, 'Edgar', 'Codd')
    (4, 'Martin', 'Forest')
```

```
cur.execute("SELECT id, first_name, last_name FROM customers WHERE id = 2")
result = cur.fetchone()
print(result)
```

```
    (2, 'Donald', 'Knuth')
```

```
con.row_factory = sqlite3.Row
cur = con.cursor()
cur.execute("SELECT id, first_name, last_name FROM customers WHERE id = 4")
result = cur.fetchone()
id, first_name, last_name = result['id'], result['first_name'], result['last_name']
print(f"Customer: {first_name} {last_name}'s id is {id}")
```

```
    Customer: Martin Forest's id is 4
```

Supplementary Activity:

1. Create a database and call it user.db
2. Create a table named "users" and insert the following: (id int, name TEXT, email TEXT)
3. Insert the following data:
   (1, 'Jonathan','jvtaylar@gmail.com'),
   (2, 'John','jonathan@gmail.com'),
   (3,'cpeEncoders','encoders@gmail.com')
4. Select all data from users.
5. Select id = 3 from users.
6. Update user id = 3 name and set it to "James."
7. Insert the following data: (4, 'Cynthia','cynthia@gmail.com')
8. Delete id = 4 from users.
9. Display all contents in a formatted way.

```
import sqlite3
import os

DP = os.path.join("user.db")
```

```
#Function for connection
#DataBaseConnection (DBC)
def DBC(dbPath = DP):
  conV2 = sqlite3.connect(dbPath)
  return conV2
```

```
conV2_1 = DBC()
curV2 = conV2_1.cursor()
```

```
client_sql = """CREATE TABLE users
                (id integer PRIMARY KEY,
                name text NOT NULL,
                email text NOT NULL)"""
curV2.execute(client_sql)


#Function for Inserting data
def InsertData(Name, email):
  User_sql = "INSERT INTO users(name, email) VALUES (?, ?)"
  curV2.execute(User_sql, (Name, email))
  conV2_1.commit()


InsertData('John', 'jonathan@gmail.com')
InsertData('cpeEncoders', 'encoders@gmail.com')


curV2.execute("SELECT * FROM users")
results = curV2.fetchall()
for db in results:
  print(db)
```

```
    (1, 'Jonathan', 'jvtaylar@gmail.com')
    (2, 'John', 'jonathan@gmail.com')
    (3, 'James', 'encoders@gmail.com')
    (4, 'Cynthia', 'Cynthia@gmail.com')
```

```
curV2.execute("SELECT id, name, email FROM users WHERE id = 3")
print(curV2.fetchone())
```

```
    (3, 'cpeEncoders', 'encoders@gmail.com')
```

```
curV2.execute("UPDATE users SET name = 'James' WHERE id = 3")
```

```
    <sqlite3.Cursor at 0x7c3572e13cc0>
```

```
curV2.execute("SELECT id, name, email FROM users WHERE id = 3")
print(curV2.fetchone())
```

```
    (3, 'James', 'encoders@gmail.com')
```

```
InsertData('Cynthia', 'Cynthia@gmail.com')


curV2.execute("SELECT * FROM users")
results = curV2.fetchall()
for db in results:
  print(db)
```

```
    (1, 'Jonathan', 'jvtaylar@gmail.com')
    (2, 'John', 'jonathan@gmail.com')
    (3, 'James', 'encoders@gmail.com')
    (4, 'Cynthia', 'Cynthia@gmail.com')
```

```
curV2.execute("SELECT id, name, email FROM users")
FormatResult = [f"{id:<5}{name:<15}{email:>5}" for id, name, email in curV2.fetchall()]
id, name, email = "Id", "Name", "email"
print('\n'.join([f"{id:<5}{name:<15}{email:>5}"] + FormatResult))
```

```
    Id   Name           email
    1    Jonathan       jvtaylar@gmail.com
    2    John           jonathan@gmail.com
    3    James          encoders@gmail.com
    4    Cynthia        Cynthia@gmail.com
```

```
curV2.execute("DELETE FROM users WHERE id = 4")
```

```
    <sqlite3.Cursor at 0x7c3572e13cc0>
```

```
curV2.execute("SELECT id, name, email FROM users")
FormatResult = [f"{id:<5}{name:<15}{email:>5}" for id, name, email in curV2.fetchall()]
id, name, email = "Id", "Name", "email"
print('\n'.join([f"{id:<5}{name:<15}{email:>5}"] + FormatResult))
```

```
Id   Name          email
1    Jonathan      jvtaylar@gmail.com
2    John          jonathan@gmail.com
3    James         encoders@gmail.com
```