Python SQLite POS Relational Database CLI

Contents

Time required: 120 minutes

SQL Tutorial

- https://www.w3schools.com/sql/sql_intro.asp
- https://www.w3schools.com/sql/sql_syntax.asp
- https://www.w3schools.com/sql/sql_create_db.asp
- https://www.w3schools.com/sql/sql_create_table.asp

- https://www.w3schools.com/sql/sql_drop_table.asp
- https://www.w3schools.com/sql/sql_insert.asp
- https://www.w3schools.com/sql/sql_update.asp
- https://www.w3schools.com/sql/sql_delete.asp
- https://www.w3schools.com/sql/sql_select.asp
- https://www.w3schools.com/sql/sql_in.asp
- https://www.w3schools.com/sql/sql_wildcards.asp
- *https://www.w3schools.com/sql/sql_join_inner.asp*

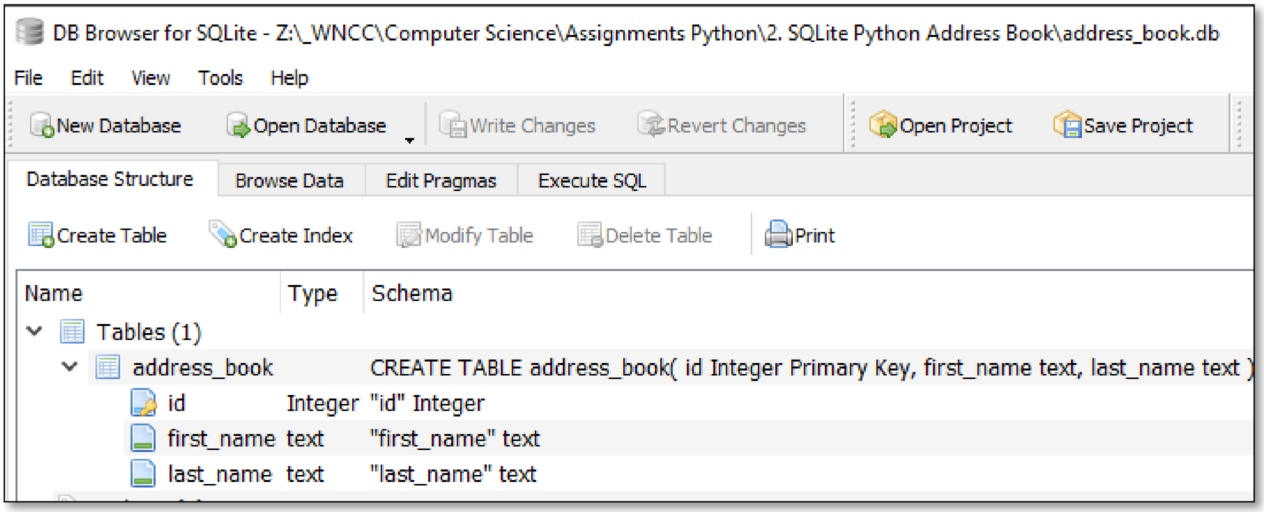
Entity Relationship Diagram Tutorials

- https://www.tutorialspoint.com/dbms/er_model_basic_concepts.htm
- https://www.tutorialspoint.com/dbms/er_diagram_representation.htm
- https://www.lucidchart.com/pages/videos/entity-relationship-diagram-erd-tutorial-part-1

SQLite Database Browser

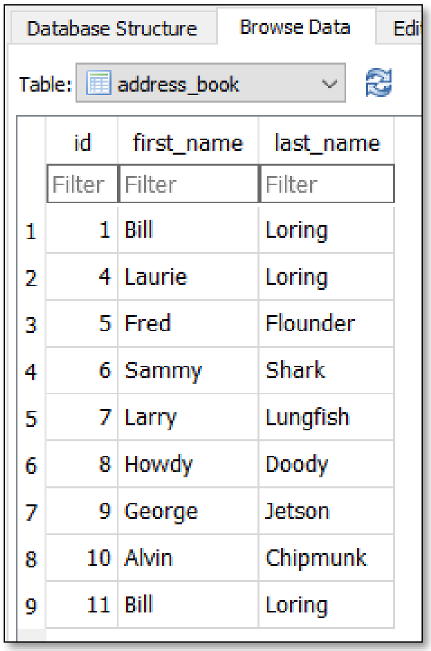
This is a handy tool to look at, troubleshoot, and manipulate your database.

1. Go to https://sqlitebrowser.org
2. Go to the **Download** tab.
3. Download the **Windows PortableApp** → **DB Browser for SQLite - PortableApp**
4. Double Click the installation file. Click **Next**.
5. Click **Install**. Click **Finish**.
6. You will find a new folder: **SQLiteDatabaseBrowserPortable**
7. This folder can be moved anywhere, the program will work just ifne.
8. In the folder you will find **SQLiteDatabaseBrowserPortable.exe**
9. Double Click the file. Click **OK** on the warning.
10. Use the **Open Database** button to open your database.



Name	Type	Schema
Tables (1)		
address_book		CREATE TABLE address_book(id Integer Primary Key, first_name text, last_name text)
id	Integer	"id" Integer
first_name	text	"first_name" text
last_name	text	"last_name" text

Click the **Browse Data** tab to see your records.



	id	first_name	last_name
	Filter	Filter	Filter
1	1	Bill	Loring
2	4	Laurie	Loring
3	5	Fred	Flounder
4	6	Sammy	Shark
5	7	Larry	Lungfish
6	8	Howdy	Doody
7	9	George	Jetson
8	10	Alvin	Chipmunk
9	11	Bill	Loring

Click the **Close Database** button when you are done.

SQLite Relational Database

SQLite is a relational database. We create tables related by primary keys. We will design our databases using an ERD (Entity Relationship Diagram). www.lucidchart.com is free web-based diagram site used in these SQLite tutorials.

In this tutorial, we will create two related tables, then 3 related tables with a bridge/junction entity.

What is an ERD?

ERD: An Entity Relationship Diagram, also known as ERD, is a diagram that displays the relationship of entity sets stored in a database. ER diagrams help to explain the logical structure of databases. ER diagrams are created based on three basic concepts: entities (tables), attributes (fields), and relationships.

ER Diagrams contain different symbols that use rectangles to represent entities, ovals to define attributes and diamond shapes to represent relationships.

At first look, an ER diagram looks very similar to the flowchart. However, ER Diagram includes many specialized symbols, and its meanings make this model unique. The purpose of ER Diagram is to represent the entity framework infrastructure.

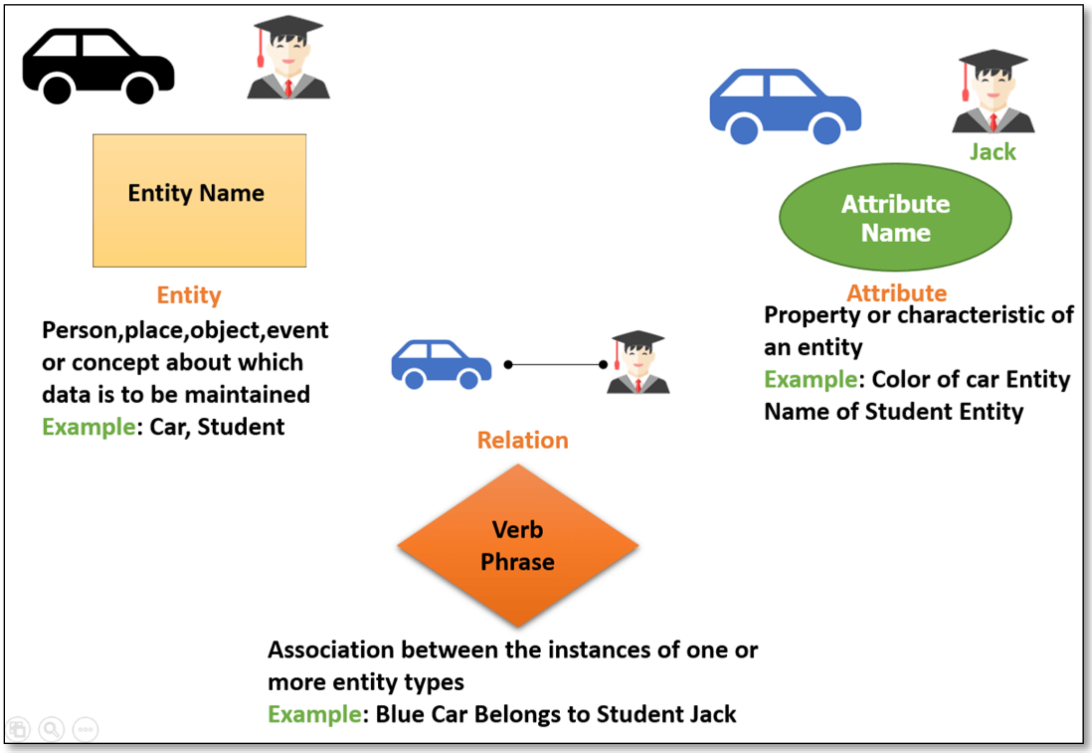
Components of the ER Diagram

This model is based on three basic concepts:

- Entities (Objects)
- Attributes (Properties)
- Relationships

ER Diagram Examples

For example, in a University database, we might have entities for Students, Courses, and Professors. The Student entity can have attributes like StudentID, Name, and DeptID. They might have relationships with Courses and Professors.


Entity Name
Entity
Person,place,object,event or concept about which data is to be maintained
Example: Car, Student
Relation
Verb Phrase
Association between the instances of one or more entity types
Example: Blue Car Belongs to Student Jack
Jack
Attribute Name
Attribute
Property or characteristic of an entity
Example: Color of car Entity Name of Student Entity

2-Table ERD

Primary Key: A primary key is a column or a set of columns in a table whose values uniquely identify a row in the table.

Foreign Key: A foreign key is a column or a set of columns in a table whose values correspond to the values of the primary key in another table.

These two tables are related through a primary key in the customer table, **cust_id**. A foreign key **cust_id** is in the product table. This key connects the two tables. This is an example of a one-to-many relationship.

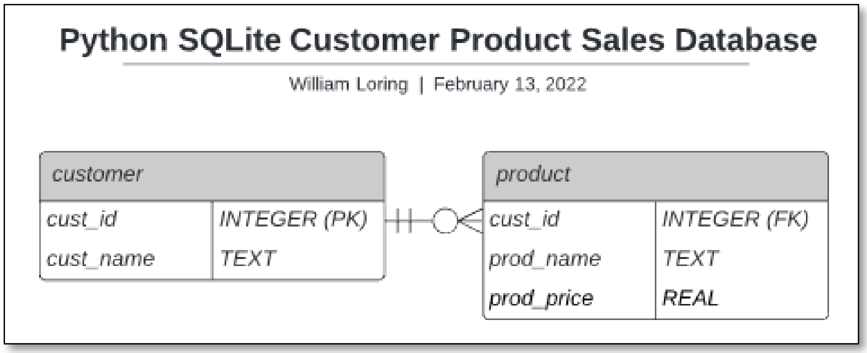
Business Rules

- A customer can purchase many products.
- A product can only have one customer.

Reference: https://vertabelo.com/blog/crow-s-foot-notation/

ERD: An Entity Relationship Diagram, also known as ERD, is a diagram that displays the relationship of entity sets stored in a database. In other words, ER diagrams help to explain the logical structure of databases. ER diagrams are created based on three basic concepts: entities (tables), attributes (fields), and relationships.

Entity Relationship Diagram of 2 table related database.

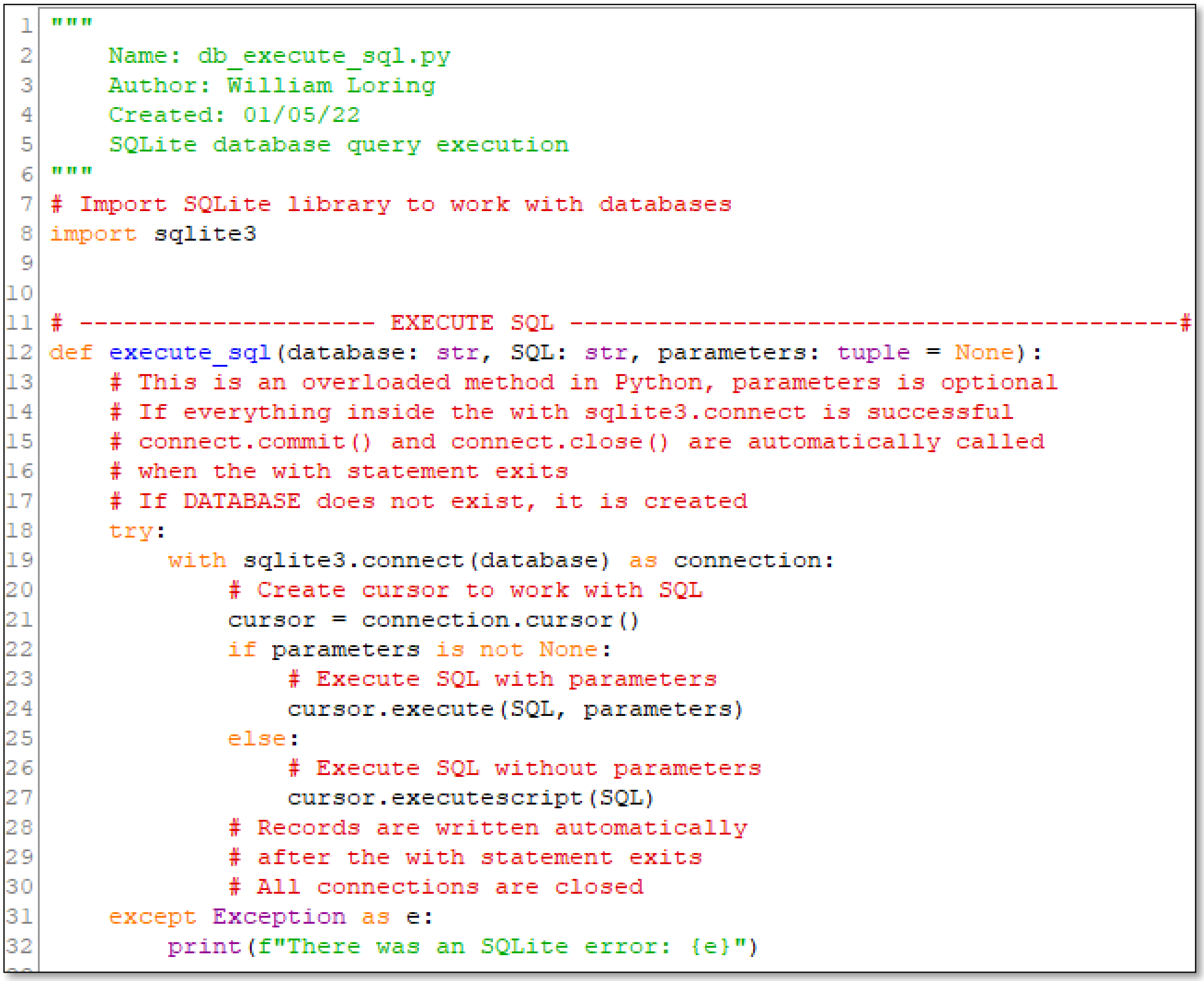

Python SQLite Customer Product Sales Database
William Loring | February 13, 2022

customer	
cust_id	INTEGER (PK)
cust_name	TEXT

product	
cust_id	INTEGER (FK)
prod_name	TEXT
prod_price	REAL

Tutorial 1: DB Execute SQL

This is the **execute_sql** method we have used in previous SQLite projects. This module will execute all SQL requests against our database. It has been moved to a separate module file for portability.



```python
"""
    Name: db_execute_sql.py
    Author: William Loring
    Created: 01/05/22
    SQLite database query execution
"""
# Import SQLite library to work with databases
import sqlite3


# -------------------- EXECUTE SQL ----------------------------------------#
def execute_sql(database: str, SQL: str, parameters: tuple = None):
    # This is an overloaded method in Python, parameters is optional
    # If everything inside the with sqlite3.connect is successful
    # connect.commit() and connect.close() are automatically called
    # when the with statement exits
    # If DATABASE does not exist, it is created
    try:
        with sqlite3.connect(database) as connection:
            # Create cursor to work with SQL
            cursor = connection.cursor()
            if parameters is not None:
                # Execute SQL with parameters
                cursor.execute(SQL, parameters)
            else:
                # Execute SQL without parameters
                cursor.executescript(SQL)
            # Records are written automatically
            # after the with statement exits
            # All connections are closed
    except Exception as e:
        print(f"There was an SQLite error: {e}")
```

Tutorial 2: Create Database and Tables

Let’s create our database and tables.

1. Database: **store_operations.db**
2. Tables (Entities): **customer**, **product**
3. Fields (Attributes)

This is the structure of our database.

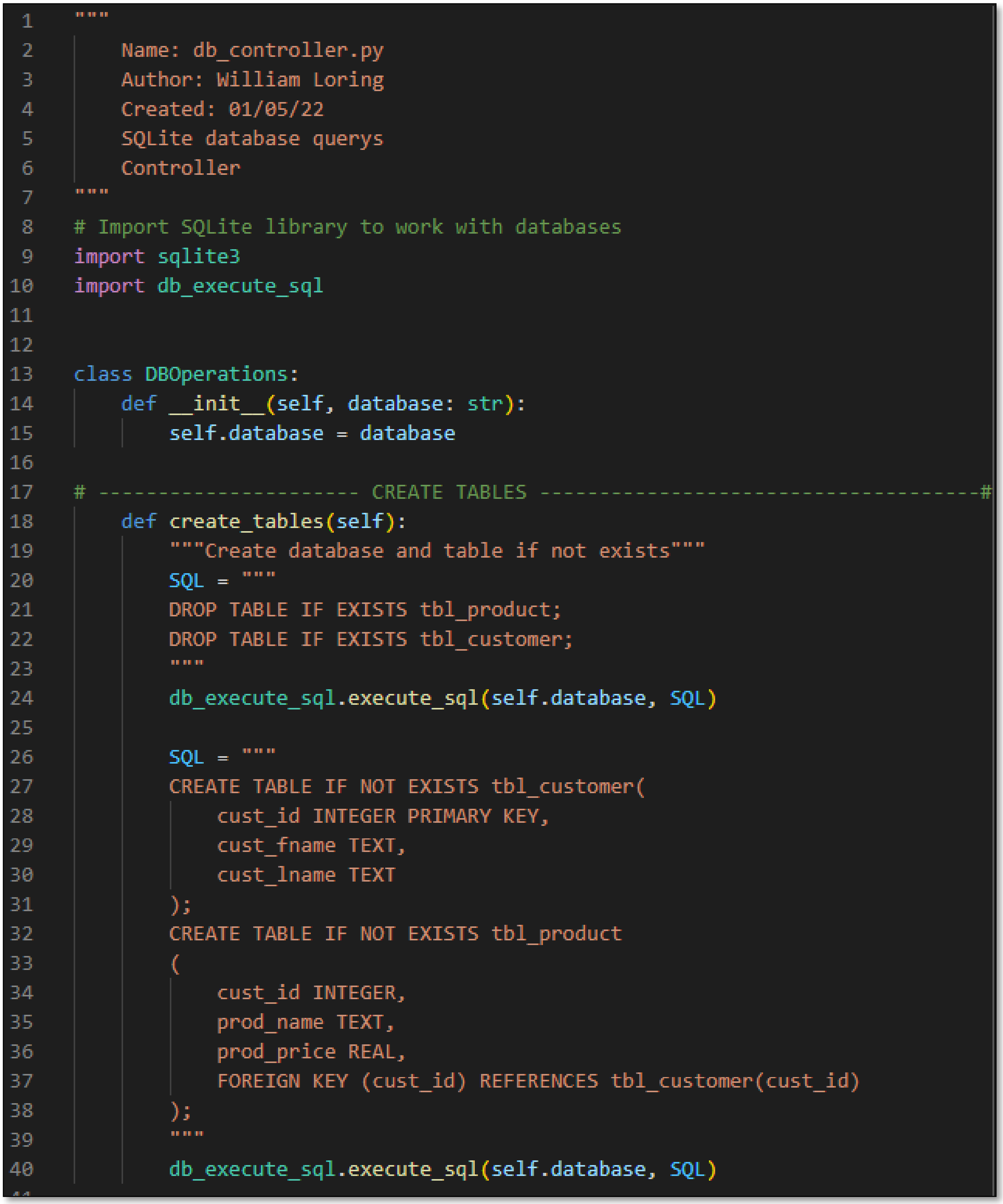
Table Name	Field Name	Field Format
Customer	cust_id	INTEGER – Primary Key
Customer	cust_name	TEXT

Product	cust_id	INTEGER – Foreign Key
Product	prod_name	TEXT
Product	prod_price	REAL

When we are developing a database, we want to be able to drop the tables to make it easier to implement changes. This allows us to start over again each time we run the program. When we are finished developing, we will comment out the drop table sql call.

The code below implements this database design and creates the related tables.

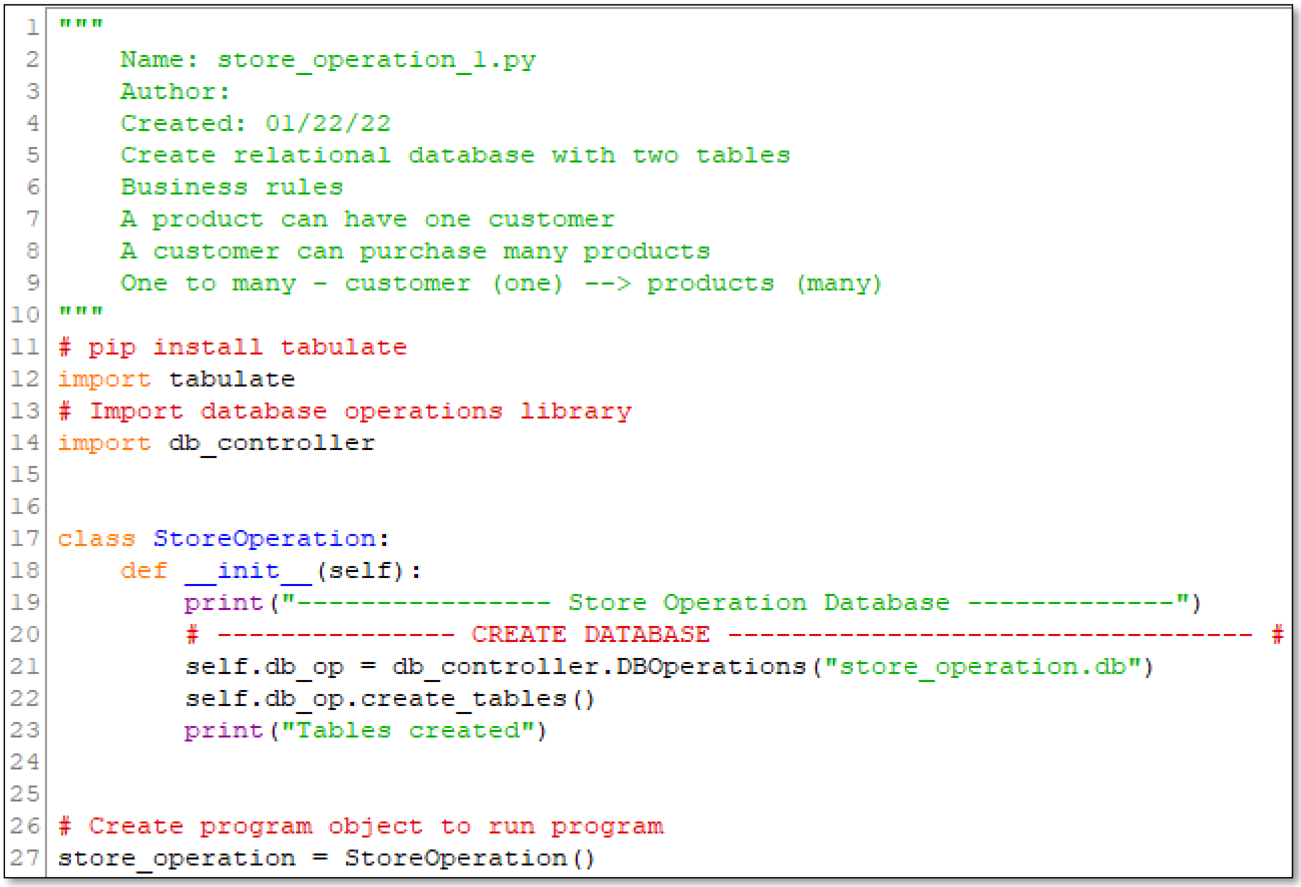
Create a Python file named **db_controller.py**



```
"""
    Name: db_controller.py
    Author: William Loring
    Created: 01/05/22
    SQLite database querys
    Controller
"""
# Import SQLite library to work with databases
import sqlite3
import db_execute_sql


class DBOperations:
    def __init__(self, database: str):
        self.database = database

# ---------------------- CREATE TABLES -------------------------------------#
    def create_tables(self):
        """Create database and table if not exists"""
        SQL = """
        DROP TABLE IF EXISTS tbl_product;
        DROP TABLE IF EXISTS tbl_customer;
        """
        db_execute_sql.execute_sql(self.database, SQL)

        SQL = """
        CREATE TABLE IF NOT EXISTS tbl_customer(
            cust_id INTEGER PRIMARY KEY,
            cust_fname TEXT,
            cust_lname TEXT
        );
        CREATE TABLE IF NOT EXISTS tbl_product
        (
            cust_id INTEGER,
            prod_name TEXT,
            prod_price REAL,
            FOREIGN KEY (cust_id) REFERENCES tbl_customer(cust_id)
        );
        """
        db_execute_sql.execute_sql(self.database, SQL)
```

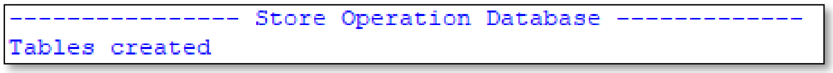
This is the beginning of our main program.

store_operation_1.py



```python
"""
    Name: store_operation_1.py
    Author:
    Created: 01/22/22
    Create relational database with two tables
    Business rules
    A product can have one customer
    A customer can purchase many products
    One to many - customer (one) --> products (many)
"""
# pip install tabulate
import tabulate
# Import database operations library
import db_controller


class StoreOperation:
    def __init__(self):
        print("---------------- Store Operation Database -------------")
        # --------------- CREATE DATABASE --------------------------------- #
        self.db_op = db_controller.DBOperations("store_operation.db")
        self.db_op.create_tables()
        print("Tables created")


# Create program object to run program
store_operation = StoreOperation()
```

Example run:



```
---------------- Store Operation Database -------------
Tables created
```

Tutorial 3: Insert Values

Let’s insert the following data into the **customer** table:

cust_id	cust_name
1	Brian
2	Sharon
3	Walter

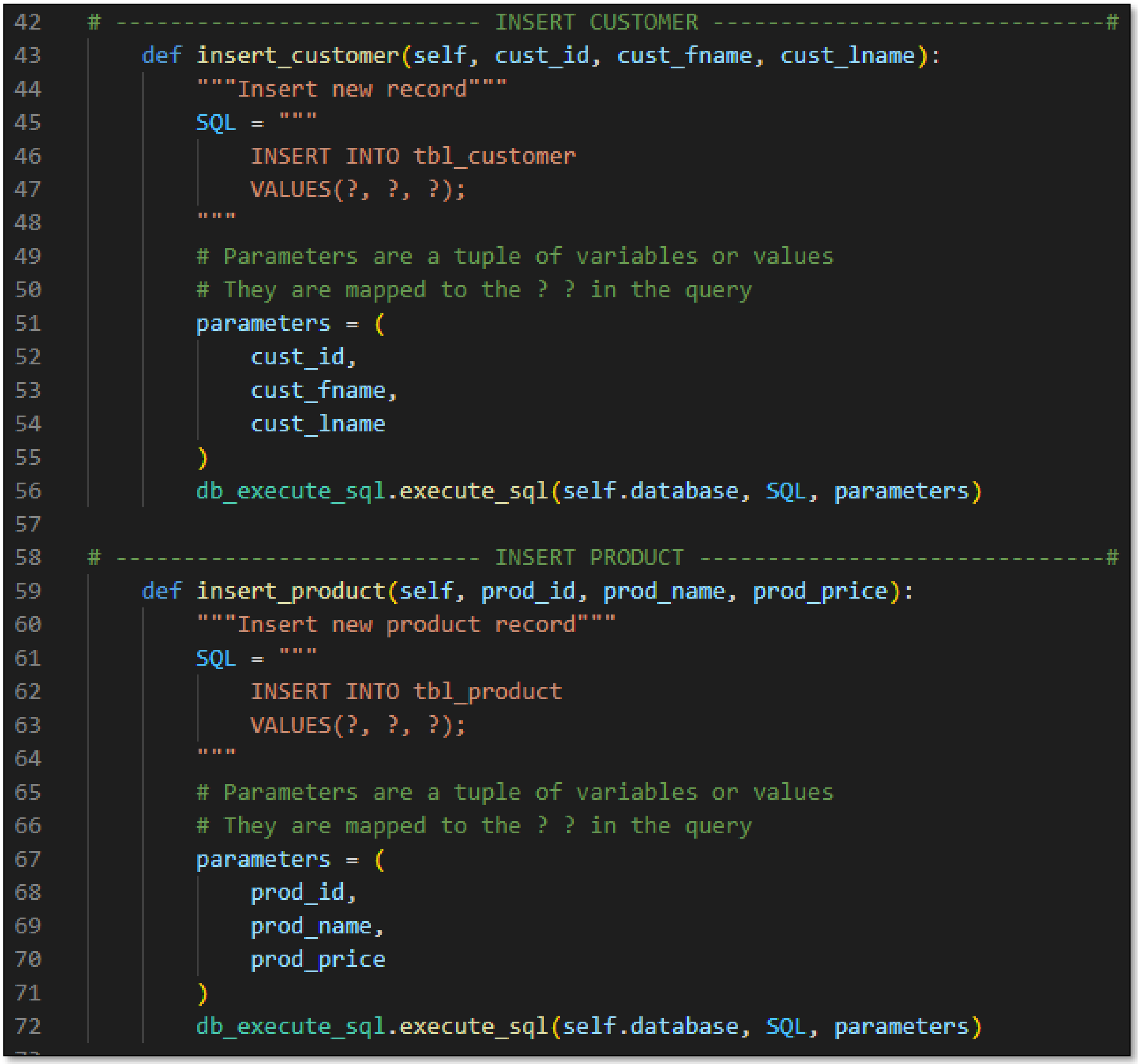
Let’s insert the following data into the **product** table:

cust_id	prod_name	prod_price
1	Windows 11	800.25

1	Windows Vista	200.99
2	iPad	300.23
3	Firewall	450.14
1	Toaster Oven	150.25

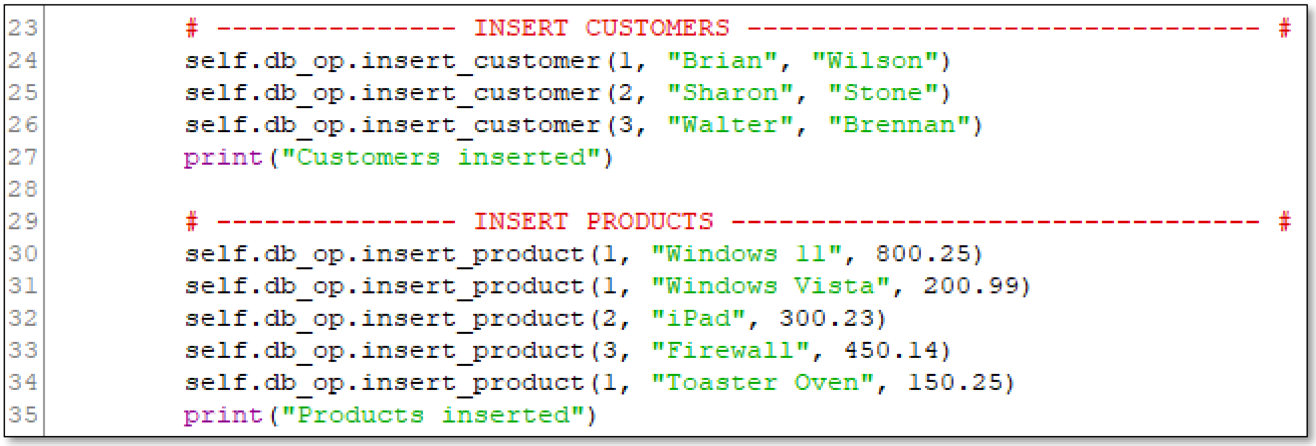
Add the following method to insert these records into our database.

db_controller.py



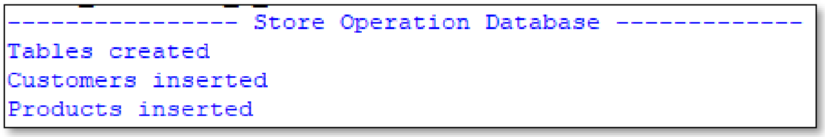
```python
# --------------------------- INSERT CUSTOMER -----------------------------#
    def insert_customer(self, cust_id, cust_fname, cust_lname):
        """Insert new record"""
        SQL = """
            INSERT INTO tbl_customer
            VALUES(?, ?, ?);
        """
        # Parameters are a tuple of variables or values
        # They are mapped to the ? ? in the query
        parameters = (
            cust_id,
            cust_fname,
            cust_lname
        )
        db_execute_sql.execute_sql(self.database, SQL, parameters)

# --------------------------- INSERT PRODUCT ------------------------------#
    def insert_product(self, prod_id, prod_name, prod_price):
        """Insert new product record"""
        SQL = """
            INSERT INTO tbl_product
            VALUES(?, ?, ?);
        """
        # Parameters are a tuple of variables or values
        # They are mapped to the ? ? in the query
        parameters = (
            prod_id,
            prod_name,
            prod_price
        )
        db_execute_sql.execute_sql(self.database, SQL, parameters)
```

store_operations_2.py



```
        # ---------------- INSERT CUSTOMERS --------------------------------- #
        self.db_op.insert_customer(1, "Brian", "Wilson")
        self.db_op.insert_customer(2, "Sharon", "Stone")
        self.db_op.insert_customer(3, "Walter", "Brennan")
        print("Customers inserted")

        # ---------------- INSERT PRODUCTS ---------------------------------- #
        self.db_op.insert_product(1, "Windows 11", 800.25)
        self.db_op.insert_product(1, "Windows Vista", 200.99)
        self.db_op.insert_product(2, "iPad", 300.23)
        self.db_op.insert_product(3, "Firewall", 450.14)
        self.db_op.insert_product(1, "Toaster Oven", 150.25)
        print("Products inserted")
```

Example run

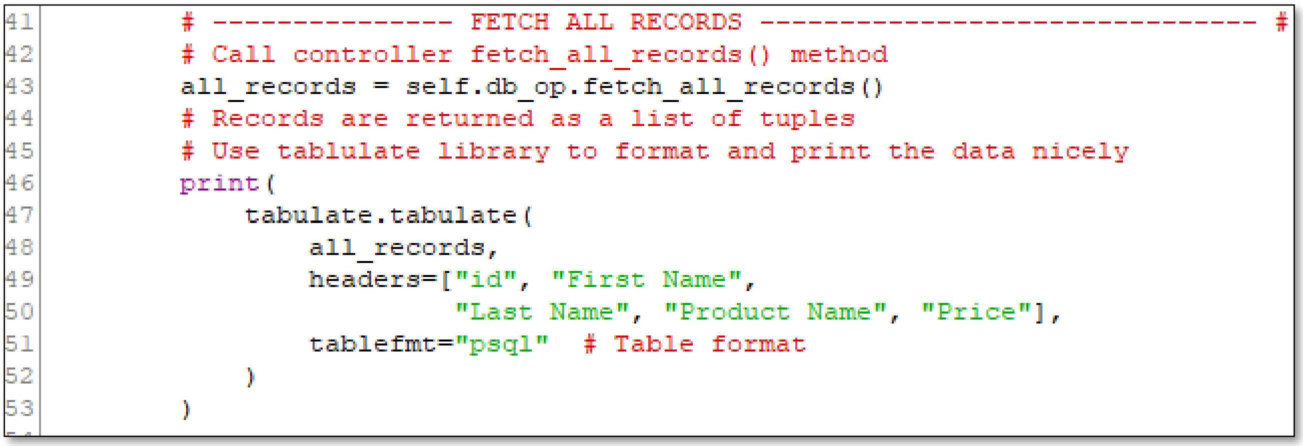


```
---------------- Store Operation Database -------------
Tables created
Customers inserted
Products inserted
```

Tutorial 4: Display the Results

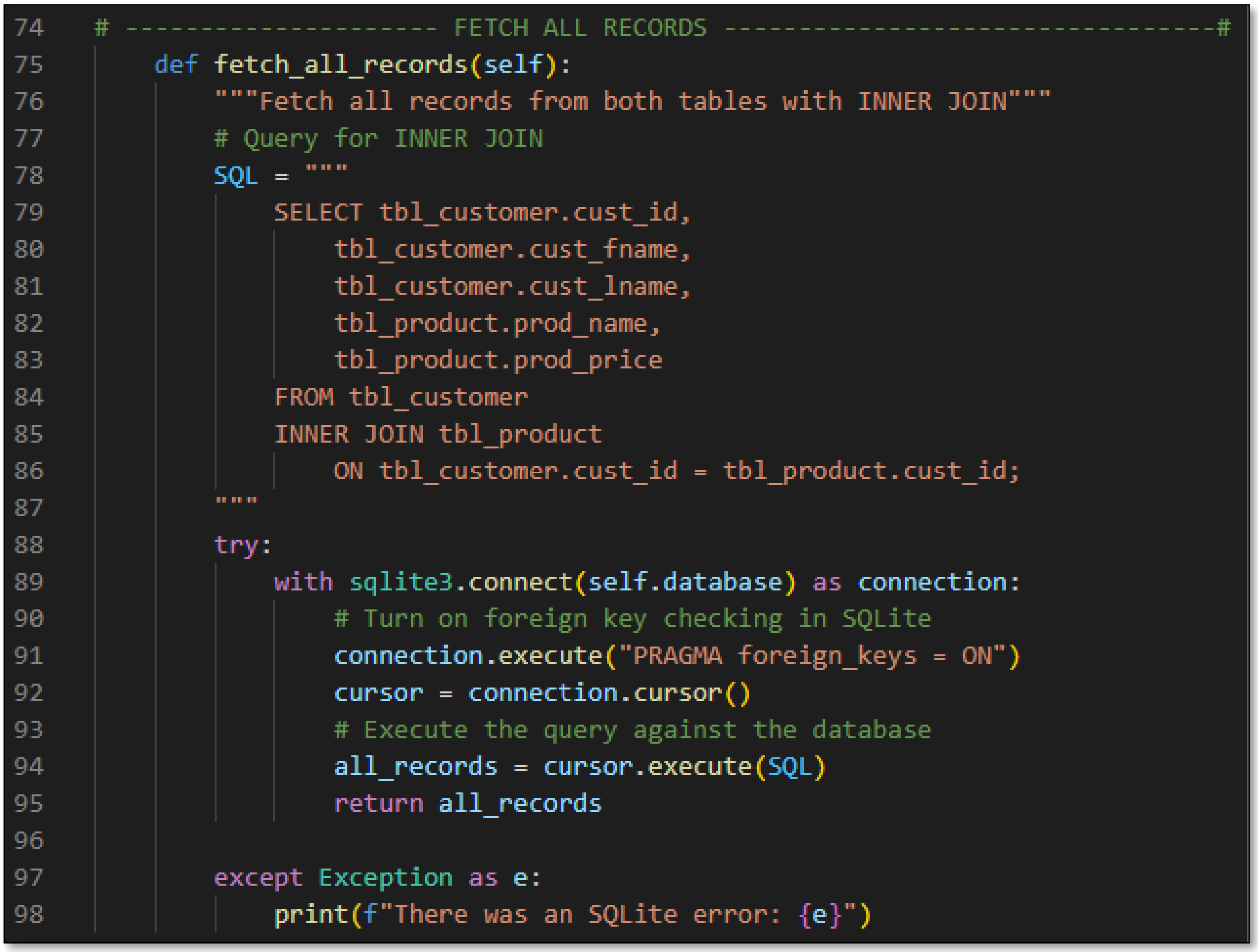
To display results from both tables, we use the INNER JOIN command using the **cust_id** PRIMARY KEY which in both tables.

Add the following method call and method to **store_operation_3.py**



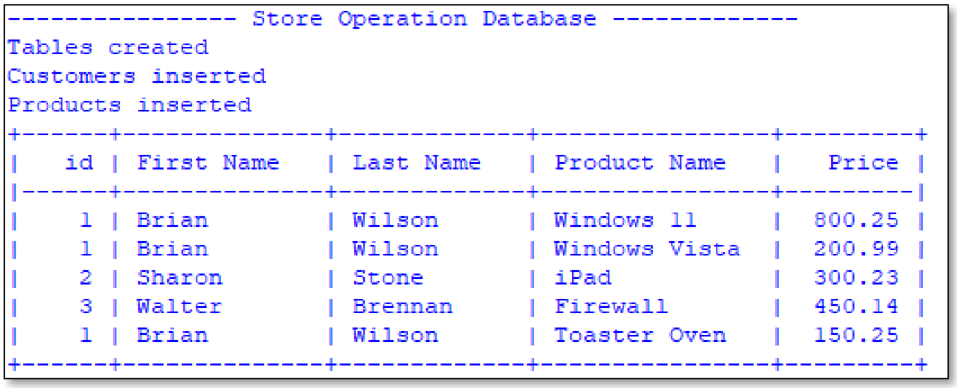
```
        # ---------------- FETCH ALL RECORDS -------------------------------- #
        # Call controller fetch_all_records() method
        all_records = self.db_op.fetch_all_records()
        # Records are returned as a list of tuples
        # Use tablulate library to format and print the data nicely
        print(
            tabulate.tabulate(
                all_records,
                headers=["id", "First Name",
                         "Last Name", "Product Name", "Price"],
                tablefmt="psql"  # Table format
            )
        )
```

db_controller.py



```python
# --------------------- FETCH ALL RECORDS ---------------------------------#
    def fetch_all_records(self):
        """Fetch all records from both tables with INNER JOIN"""
        # Query for INNER JOIN
        SQL = """
            SELECT tbl_customer.cust_id,
                tbl_customer.cust_fname,
                tbl_customer.cust_lname,
                tbl_product.prod_name,
                tbl_product.prod_price
            FROM tbl_customer
            INNER JOIN tbl_product
                ON tbl_customer.cust_id = tbl_product.cust_id;
        """

        try:
            with sqlite3.connect(self.database) as connection:
                # Turn on foreign key checking in SQLite
                connection.execute("PRAGMA foreign_keys = ON")
                cursor = connection.cursor()
                # Execute the query against the database
                all_records = cursor.execute(SQL)
                return all_records

        except Exception as e:
            print(f"There was an SQLite error: {e}")
```

Example run:



```
--------------- Store Operation Database -------------
Tables created
Customers inserted
Products inserted
+------+--------------+-------------+----------------+---------+
|   id | First Name   | Last Name   | Product Name   |   Price |
|------+--------------+-------------+----------------+---------|
|    1 | Brian        | Wilson      | Windows 11     |  800.25 |
|    1 | Brian        | Wilson      | Windows Vista  |  200.99 |
|    2 | Sharon       | Stone       | iPad           |  300.23 |
|    3 | Walter       | Brennan     | Firewall       |  450.14 |
|    1 | Brian        | Wilson      | Toaster Oven   |  150.25 |
+------+--------------+-------------+----------------+---------+
```

2-Table ERD with Bridge Entity

This is where database planning starts.

We have our entities. Like OOP, entities represent something in the real world we want to keep track of.

- Customer
- Product

Business Rules

Business rules are how the entities interact. A functional real-life customer sales tracking database would have these business rules.

- A product can be sold to many customers.
- A customer can purchase many products.

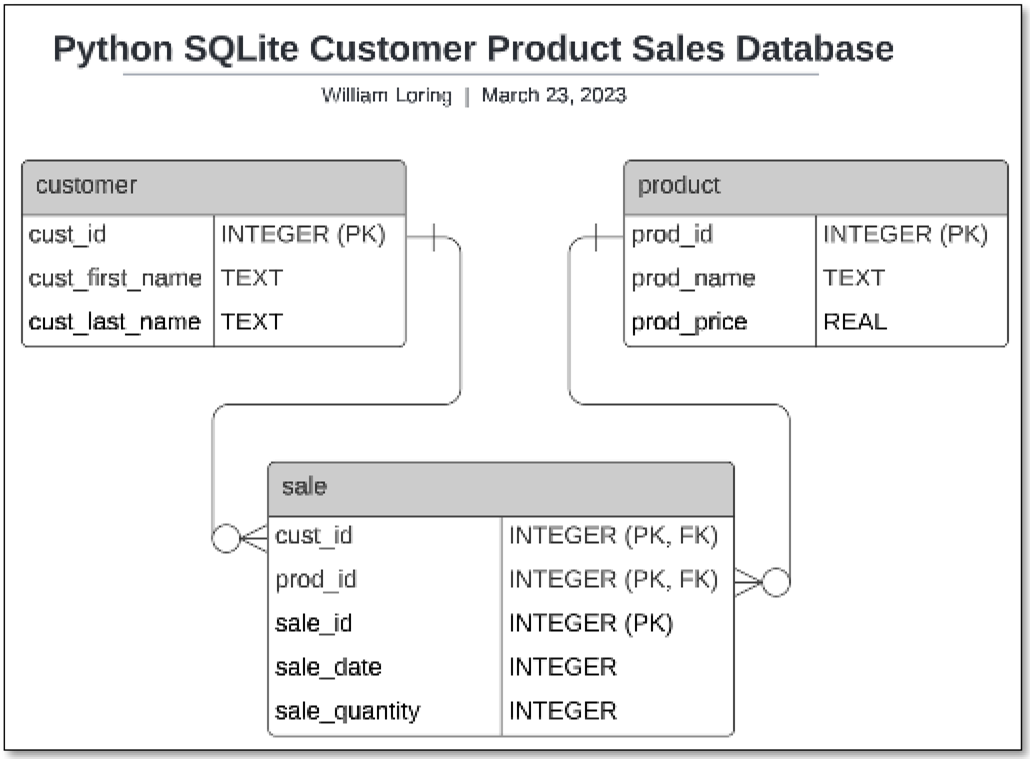
This is an example of many to many relationships. You can’t have many to many relationships in SQL. You can have two tables with a bridge or junction table connecting them as shown below to implement the many to many business rules.

Primary Key: A primary key is a column or a set of columns in a table whose values uniquely identify a row in the table.

Foreign Key: A foreign key is a column or a set of columns in a table whose values correspond to the values of the primary key in another table.

Composite Key: A composite key is made by the combination of two or more columns in a table that can be used to uniquely identify each row in the table when the columns are combined uniqueness of a row is guaranteed, but when it is taken individually it does not guarantee uniqueness, or it can also be understood as a primary key made by the combination of two or more attributes to uniquely identify every row in a table.

These two tables are related through a composite primary key in the sale table. This composite primary key connects the two tables.



Python SQLite Customer Product Sales Database

William Loring | March 23, 2023

customer	
cust_id	INTEGER (PK)
cust_first_name	TEXT
cust_last_name	TEXT

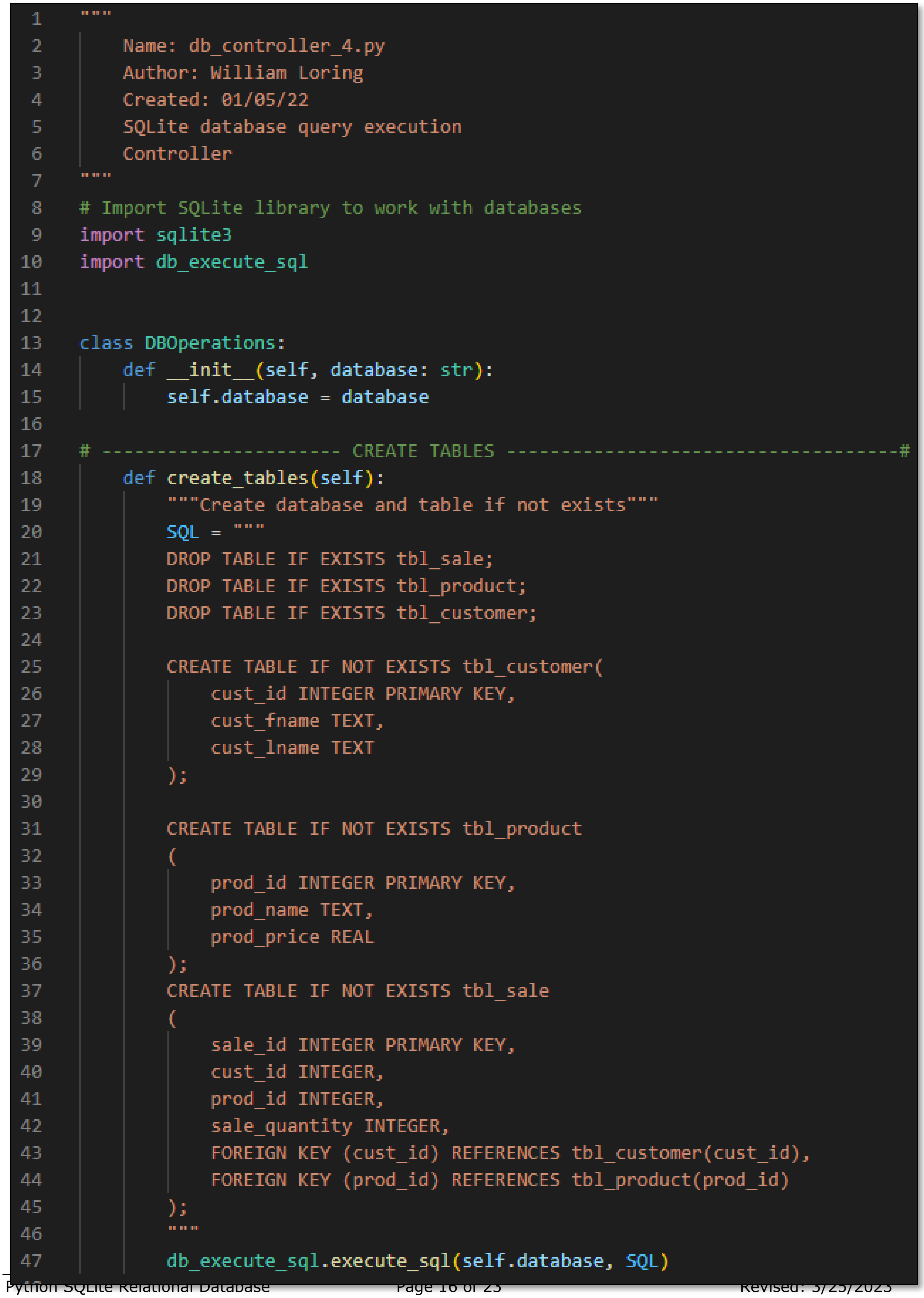
product	
prod_id	INTEGER (PK)
prod_name	TEXT
prod_price	REAL

sale	
cust_id	INTEGER (PK, FK)
prod_id	INTEGER (PK, FK)
sale_id	INTEGER (PK)
sale_date	INTEGER
sale_quantity	INTEGER

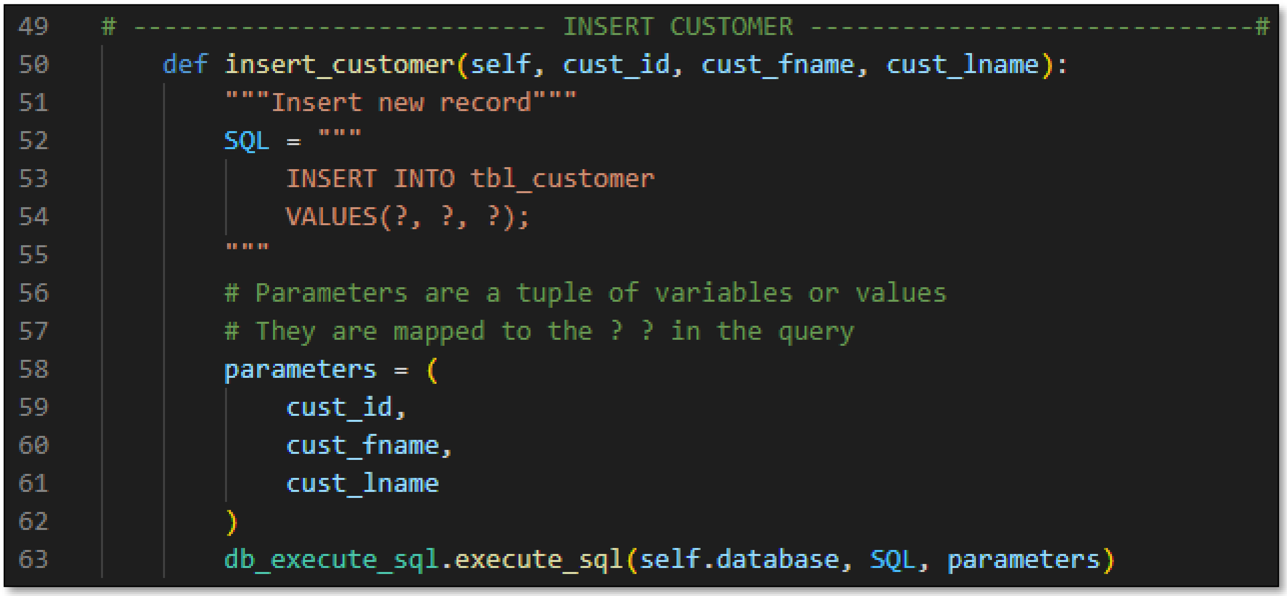
Tutorial 5: Update Controller for Sale (Bridge) Table

Copy the current tutorial to a new folder. Make the following modifications to implement the ERD above. If the method isn’t listed, it doesn’t change.

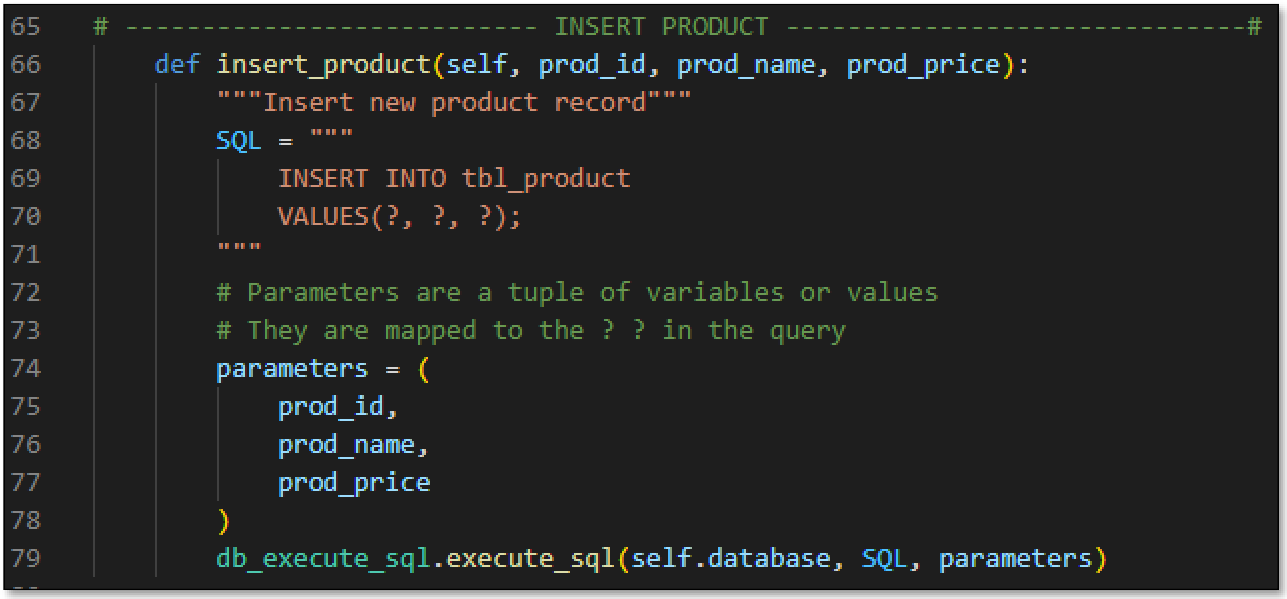
db_controller.py



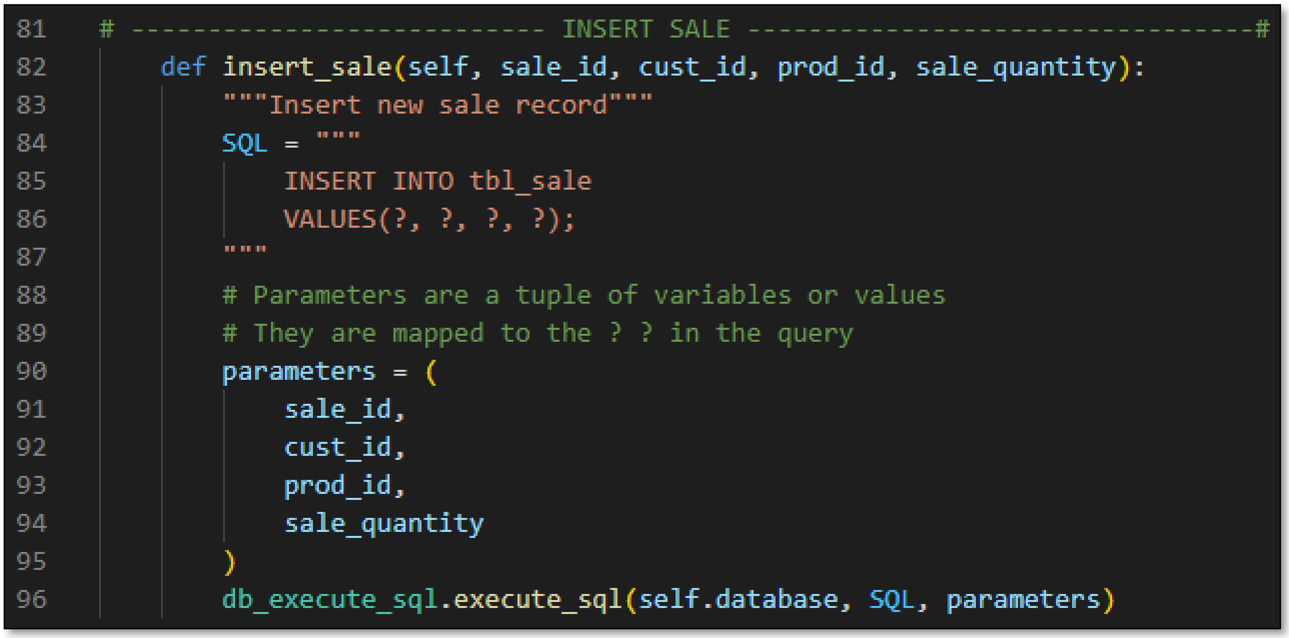
```python
"""
    Name: db_controller_4.py
    Author: William Loring
    Created: 01/05/22
    SQLite database query execution
    Controller
"""
# Import SQLite library to work with databases
import sqlite3
import db_execute_sql


class DBOperations:
    def __init__(self, database: str):
        self.database = database

# ---------------------- CREATE TABLES ----------------------------------#
    def create_tables(self):
        """Create database and table if not exists"""
        SQL = """
        DROP TABLE IF EXISTS tbl_sale;
        DROP TABLE IF EXISTS tbl_product;
        DROP TABLE IF EXISTS tbl_customer;

        CREATE TABLE IF NOT EXISTS tbl_customer(
            cust_id INTEGER PRIMARY KEY,
            cust_fname TEXT,
            cust_lname TEXT
        );

        CREATE TABLE IF NOT EXISTS tbl_product
        (
            prod_id INTEGER PRIMARY KEY,
            prod_name TEXT,
            prod_price REAL
        );
        CREATE TABLE IF NOT EXISTS tbl_sale
        (
            sale_id INTEGER PRIMARY KEY,
            cust_id INTEGER,
            prod_id INTEGER,
            sale_quantity INTEGER,
            FOREIGN KEY (cust_id) REFERENCES tbl_customer(cust_id),
            FOREIGN KEY (prod_id) REFERENCES tbl_product(prod_id)
        );
        """
        db_execute_sql.execute_sql(self.database, SQL)
```



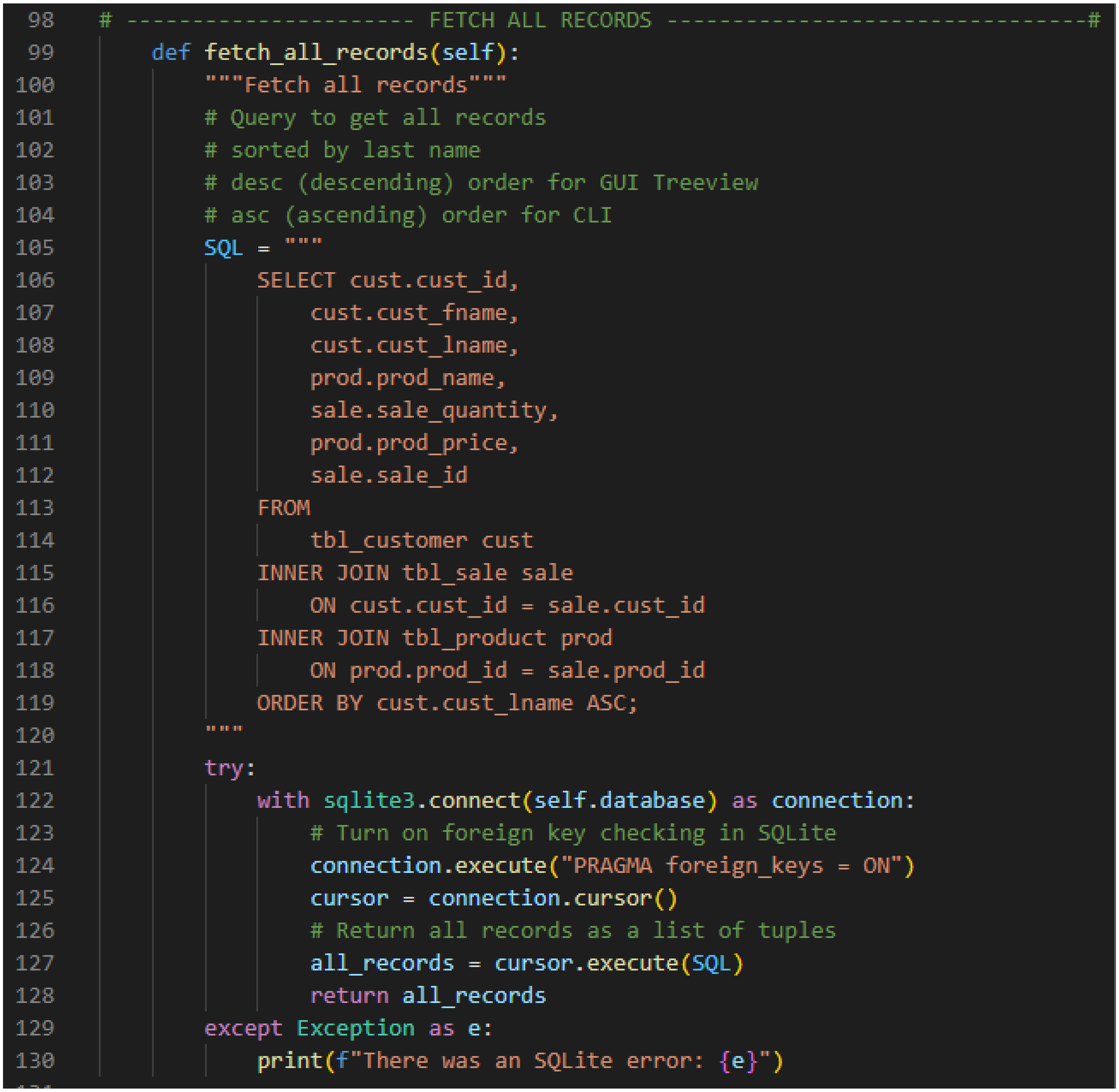
```python
# --------------------------- INSERT CUSTOMER -----------------------------#
    def insert_customer(self, cust_id, cust_fname, cust_lname):
        """Insert new record"""
        SQL = """
            INSERT INTO tbl_customer
            VALUES(?, ?, ?);
        """
        # Parameters are a tuple of variables or values
        # They are mapped to the ? ? in the query
        parameters = (
            cust_id,
            cust_fname,
            cust_lname
        )
        db_execute_sql.execute_sql(self.database, SQL, parameters)
```



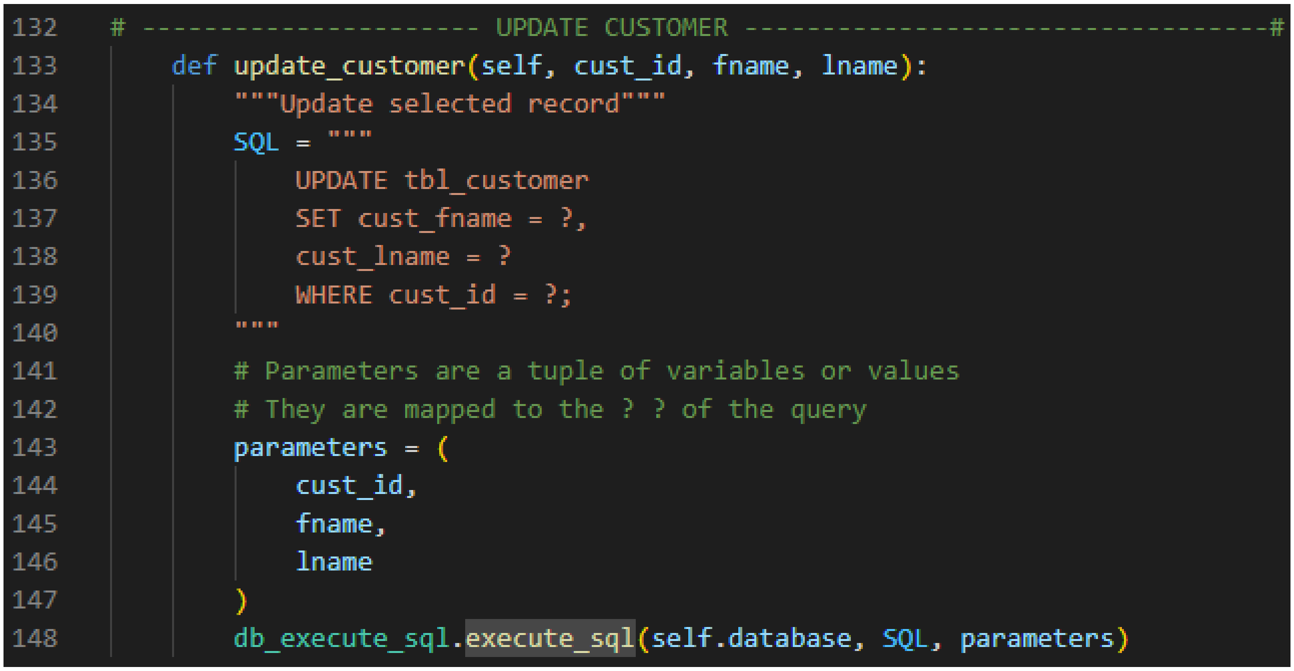
```python
# --------------------------- INSERT PRODUCT ------------------------------#
    def insert_product(self, prod_id, prod_name, prod_price):
        """Insert new product record"""
        SQL = """
            INSERT INTO tbl_product
            VALUES(?, ?, ?);
        """
        # Parameters are a tuple of variables or values
        # They are mapped to the ? ? in the query
        parameters = (
            prod_id,
            prod_name,
            prod_price
        )
        db_execute_sql.execute_sql(self.database, SQL, parameters)
```



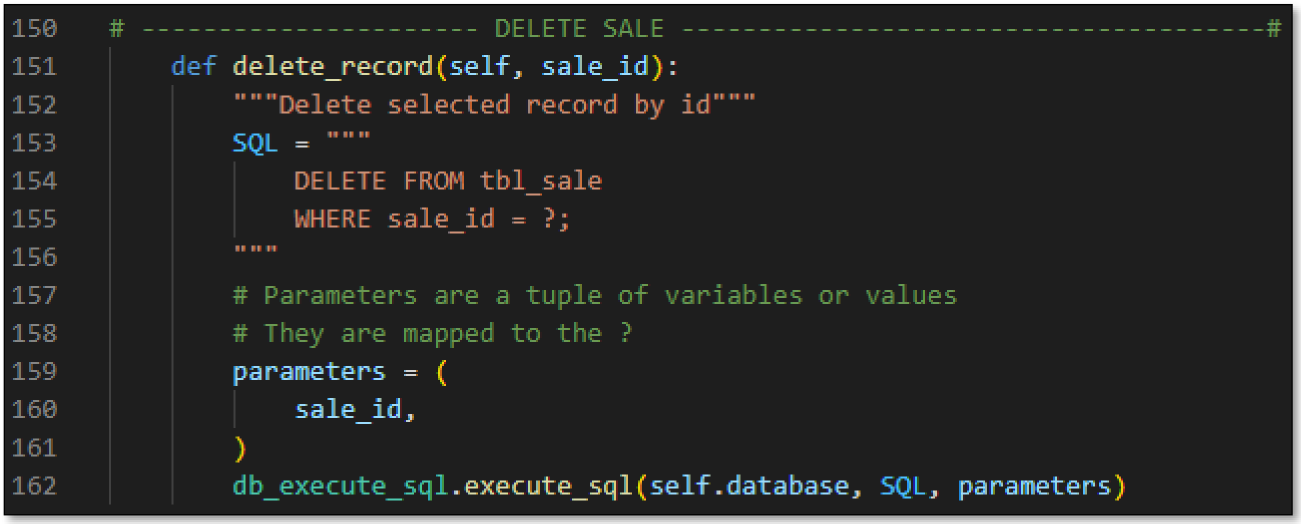
```python
# -------------------------- INSERT SALE --------------------------------#
    def insert_sale(self, sale_id, cust_id, prod_id, sale_quantity):
        """Insert new sale record"""
        SQL = """
            INSERT INTO tbl_sale
            VALUES(?, ?, ?, ?);
        """
        # Parameters are a tuple of variables or values
        # They are mapped to the ? ? in the query
        parameters = (
            sale_id,
            cust_id,
            prod_id,
            sale_quantity
        )
        db_execute_sql.execute_sql(self.database, SQL, parameters)
```



```python
# ---------------------- FETCH ALL RECORDS --------------------------------#
    def fetch_all_records(self):
        """Fetch all records"""
        # Query to get all records
        # sorted by last name
        # desc (descending) order for GUI Treeview
        # asc (ascending) order for CLI
        SQL = """
            SELECT cust.cust_id,
                cust.cust_fname,
                cust.cust_lname,
                prod.prod_name,
                sale.sale_quantity,
                prod.prod_price,
                sale.sale_id
            FROM
                tbl_customer cust
            INNER JOIN tbl_sale sale
                ON cust.cust_id = sale.cust_id
            INNER JOIN tbl_product prod
                ON prod.prod_id = sale.prod_id
            ORDER BY cust.cust_lname ASC;
        """
        try:
            with sqlite3.connect(self.database) as connection:
                # Turn on foreign key checking in SQLite
                connection.execute("PRAGMA foreign_keys = ON")
                cursor = connection.cursor()
                # Return all records as a list of tuples
                all_records = cursor.execute(SQL)
                return all_records
        except Exception as e:
            print(f"There was an SQLite error: {e}")
```

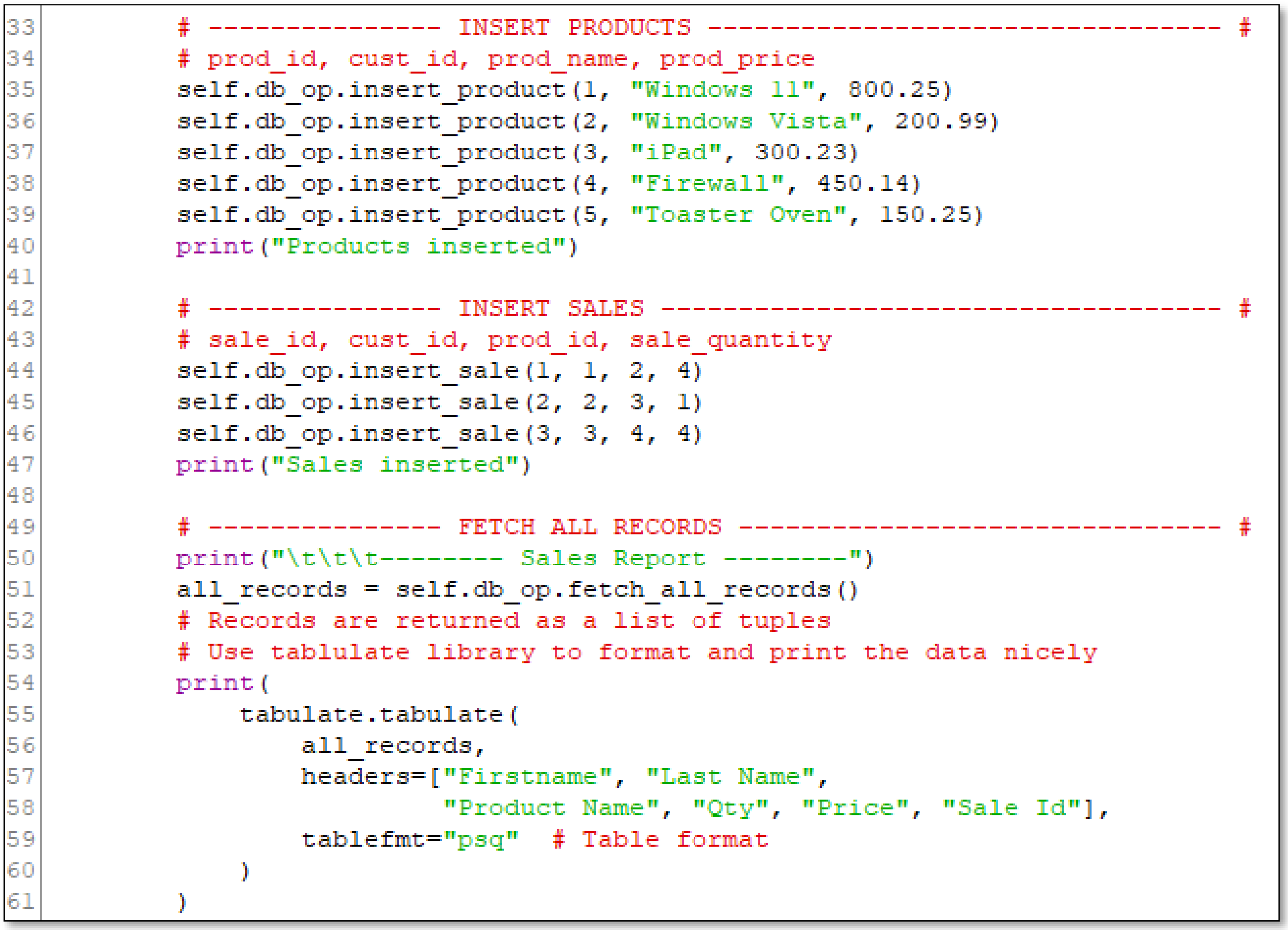


```python
# ---------------------- UPDATE CUSTOMER ---------------------------------#
    def update_customer(self, cust_id, fname, lname):
        """Update selected record"""
        SQL = """
            UPDATE tbl_customer
            SET cust_fname = ?,
            cust_lname = ?
            WHERE cust_id = ?;
        """
        # Parameters are a tuple of variables or values
        # They are mapped to the ? ? of the query
        parameters = (
            cust_id,
            fname,
            lname
        )
        db_execute_sql.execute_sql(self.database, SQL, parameters)
```



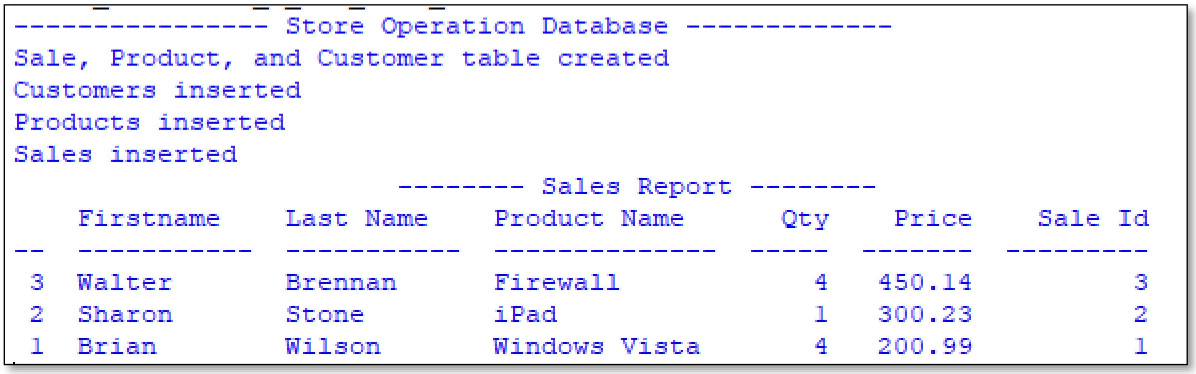
```python
# ---------------------- DELETE SALE -------------------------------------#
    def delete_record(self, sale_id):
        """Delete selected record by id"""
        SQL = """
            DELETE FROM tbl_sale
            WHERE sale_id = ?;
        """
        # Parameters are a tuple of variables or values
        # They are mapped to the ?
        parameters = (
            sale_id,
        )
        db_execute_sql.execute_sql(self.database, SQL, parameters)
```

store_operation.py changes.



```
        # ---------------- INSERT PRODUCTS ---------------------------------- #
        # prod_id, cust_id, prod_name, prod_price
        self.db_op.insert_product(1, "Windows 11", 800.25)
        self.db_op.insert_product(2, "Windows Vista", 200.99)
        self.db_op.insert_product(3, "iPad", 300.23)
        self.db_op.insert_product(4, "Firewall", 450.14)
        self.db_op.insert_product(5, "Toaster Oven", 150.25)
        print("Products inserted")

        # ---------------- INSERT SALES ------------------------------------- #
        # sale_id, cust_id, prod_id, sale_quantity
        self.db_op.insert_sale(1, 1, 2, 4)
        self.db_op.insert_sale(2, 2, 3, 1)
        self.db_op.insert_sale(3, 3, 4, 4)
        print("Sales inserted")

        # ---------------- FETCH ALL RECORDS -------------------------------- #
        print("\t\t\t--------- Sales Report ---------")
        all_records = self.db_op.fetch_all_records()
        # Records are returned as a list of tuples
        # Use tablulate library to format and print the data nicely
        print(
            tabulate.tabulate(
                all_records,
                headers=["Firstname", "Last Name",
                         "Product Name", "Qty", "Price", "Sale Id"],
                tablefmt="psq"  # Table format
            )
        )
```

Example run:

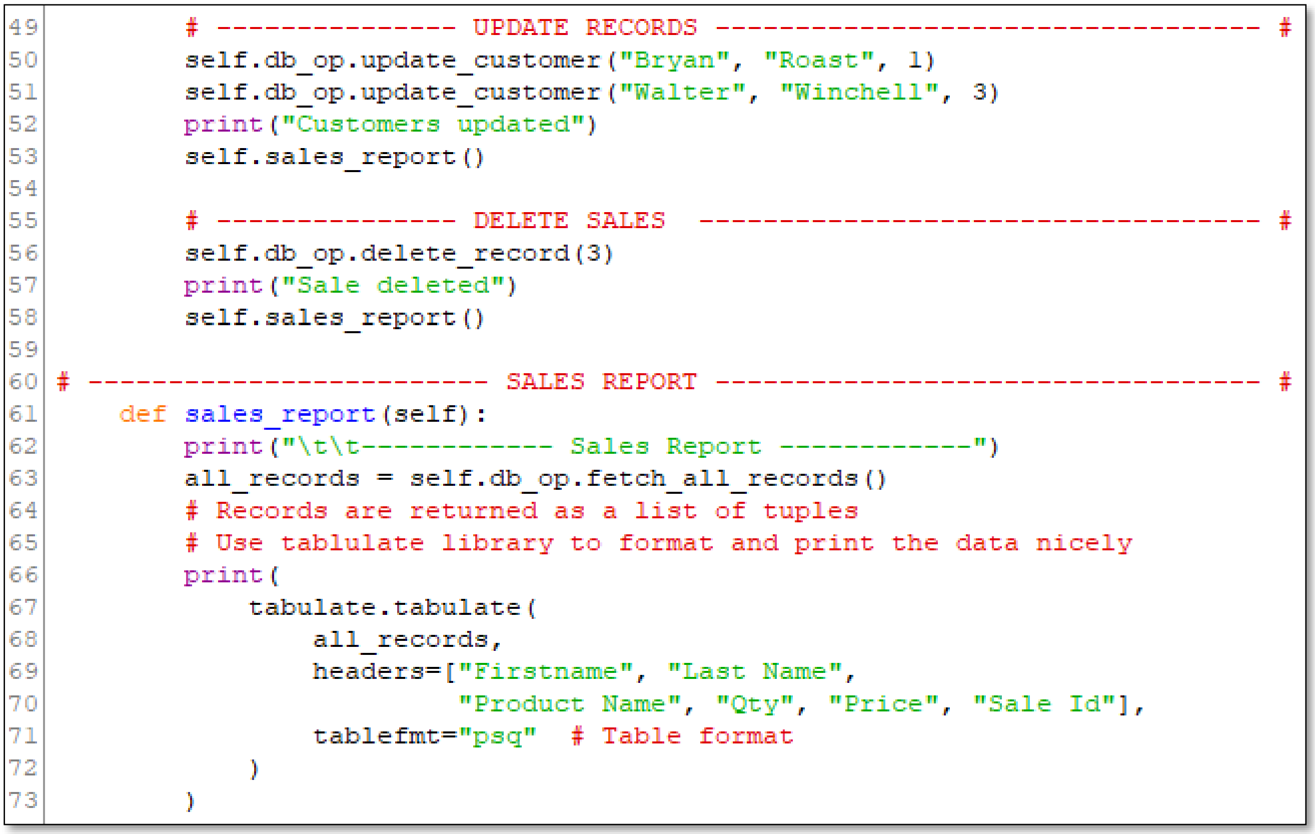


```
---------------- Store Operation Database -------------
Sale, Product, and Customer table created
Customers inserted
Products inserted
Sales inserted
                        --------- Sales Report ---------
    Firstname    Last Name    Product Name      Qty    Price    Sale Id
--  -----------  -----------  --------------  -----  -------  ---------
 3  Walter       Brennan      Firewall            4   450.14          3
 2  Sharon       Stone        iPad                1   300.23          2
 1  Brian        Wilson       Windows Vista       4   200.99          1
```

Tutorial 6: Modify and Delete Records

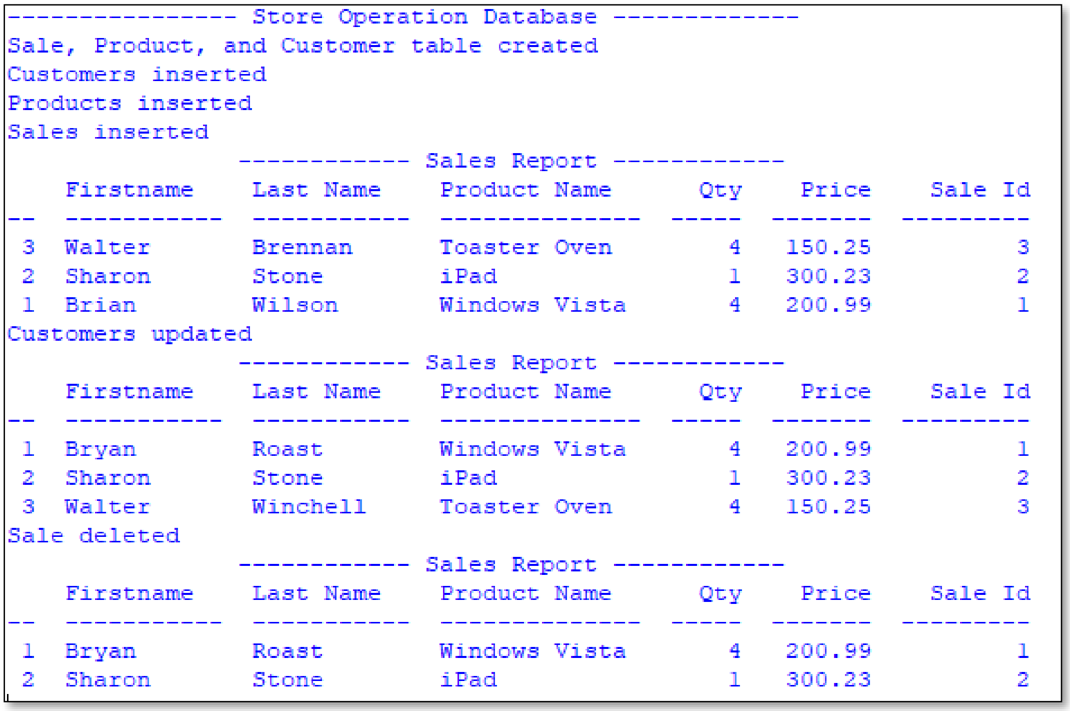
store_operation.py

Added modify of customer records, abstracted the report into its own method.



```python
        # --------------- UPDATE RECORDS ---------------------------------- #
        self.db_op.update_customer("Bryan", "Roast", 1)
        self.db_op.update_customer("Walter", "Winchell", 3)
        print("Customers updated")
        self.sales_report()

        # --------------- DELETE SALES  ----------------------------------- #
        self.db_op.delete_record(3)
        print("Sale deleted")
        self.sales_report()

# ------------------------- SALES REPORT ---------------------------------- #
    def sales_report(self):
        print("\t\t------------ Sales Report ------------")
        all_records = self.db_op.fetch_all_records()
        # Records are returned as a list of tuples
        # Use tablulate library to format and print the data nicely
        print(
            tabulate.tabulate(
                all_records,
                headers=["Firstname", "Last Name",
                         "Product Name", "Qty", "Price", "Sale Id"],
                tablefmt="psq"  # Table format
            )
        )
```

Example run:



```
---------------- Store Operation Database -------------
Sale, Product, and Customer table created
Customers inserted
Products inserted
Sales inserted
                ------------ Sales Report ------------
    Firstname    Last Name    Product Name      Qty    Price    Sale Id
--  -----------  -----------  --------------  -----  -------  ---------
 3  Walter       Brennan      Toaster Oven        4   150.25          3
 2  Sharon       Stone        iPad                1   300.23          2
 1  Brian        Wilson       Windows Vista       4   200.99          1
Customers updated
                ------------ Sales Report ------------
    Firstname    Last Name    Product Name      Qty    Price    Sale Id
--  -----------  -----------  --------------  -----  -------  ---------
 1  Bryan        Roast        Windows Vista       4   200.99          1
 2  Sharon       Stone        iPad                1   300.23          2
 3  Walter       Winchell     Toaster Oven        4   150.25          3
Sale deleted
                ------------ Sales Report ------------
    Firstname    Last Name    Product Name      Qty    Price    Sale Id
--  -----------  -----------  --------------  -----  -------  ---------
 1  Bryan        Roast        Windows Vista       4   200.99          1
 2  Sharon       Stone        iPad                1   300.23          2
```

Assignment Submission

1. Attach the program files.
2. Attach screenshots showing the successful operation of the program.
3. Submit in Blackboard.