Java 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*

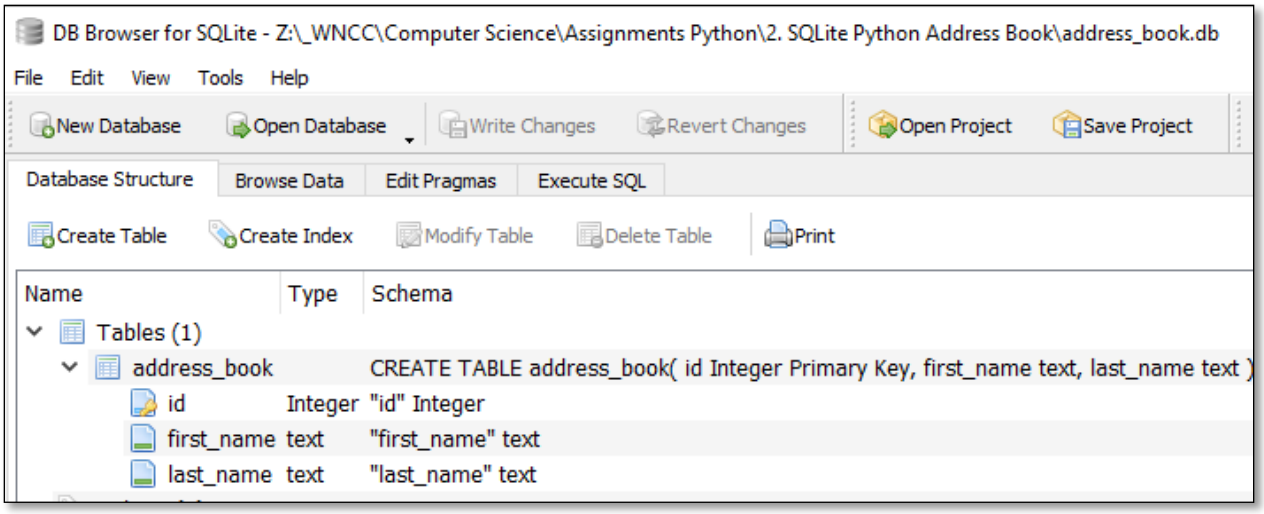
Entity Relationship Diagram Tutorials

- 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

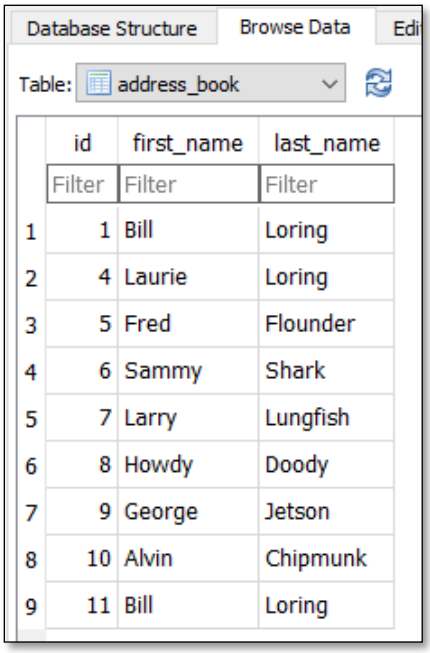
SQLite Database Browser

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.



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



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.

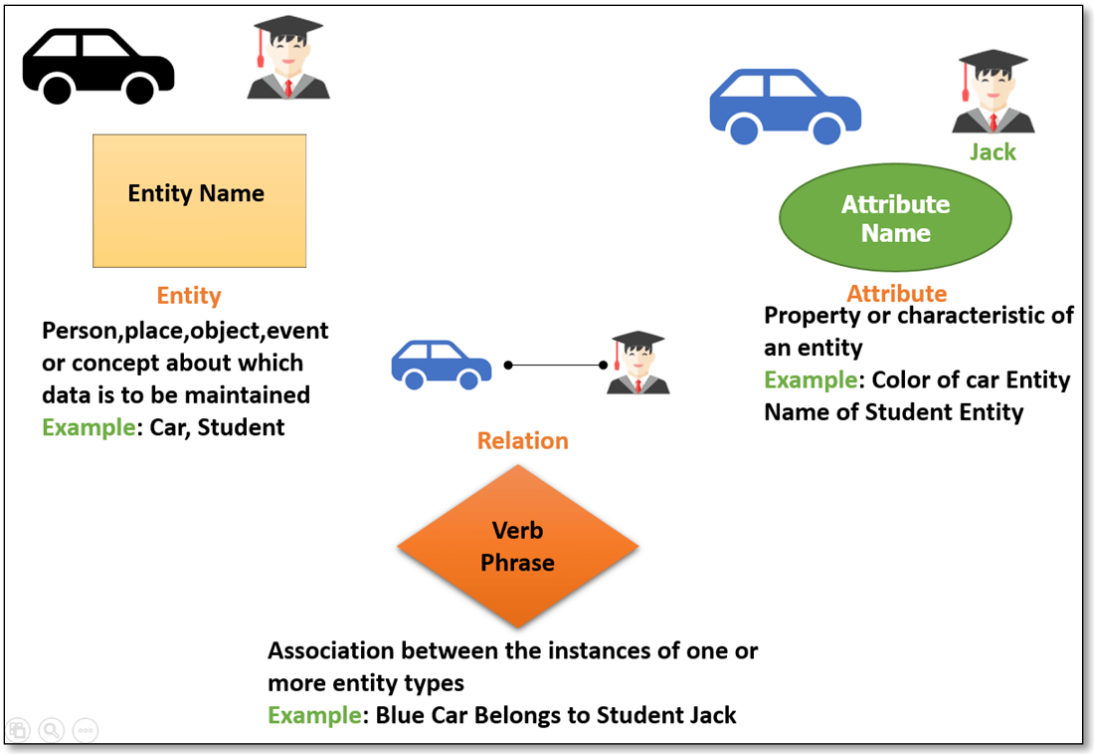
Components of the ER Diagram

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.



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.

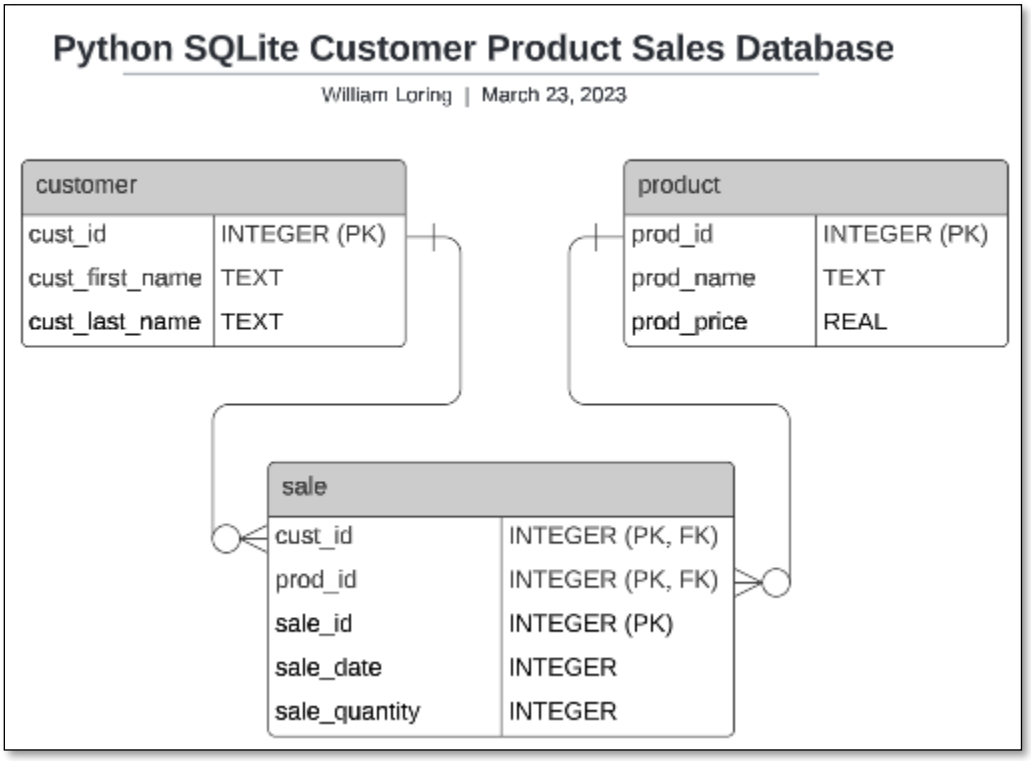
This is an example of many to many relationships. You can't have many to many relationships in SQL. You can have two tables with a bridge or junction table connecting them as shown below to implement the many to many business rules.

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.

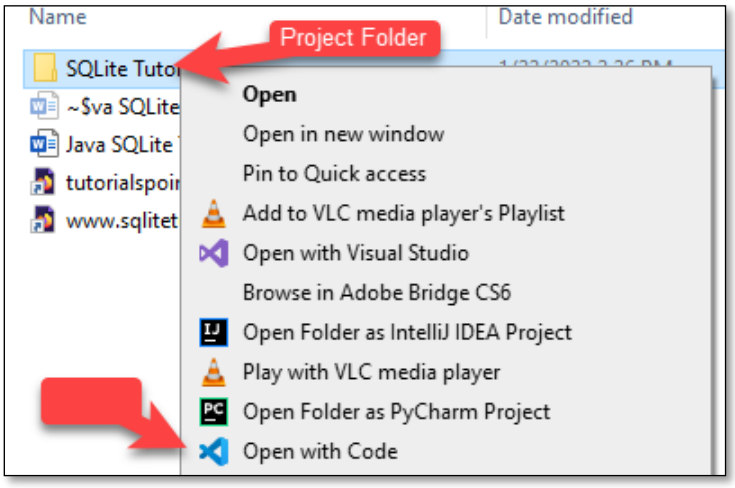


Tutorial 1: Setup the Project: JDBC and SQLite with Java

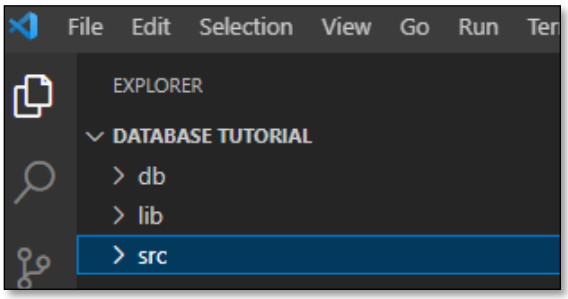
We are going to connect with our SQLite database using Java JDBC (Java Database Connectivity) API. We will be using a JDBC driver call the SQLiteJDBC Package. The SQLiteJDBC package contains both Java classes, as well as native SQLite libraries for Windows, Mac OS X, and Linux.

When we connect to an SQLite database, we are accessing data that ultimately resides in a file on our computer.

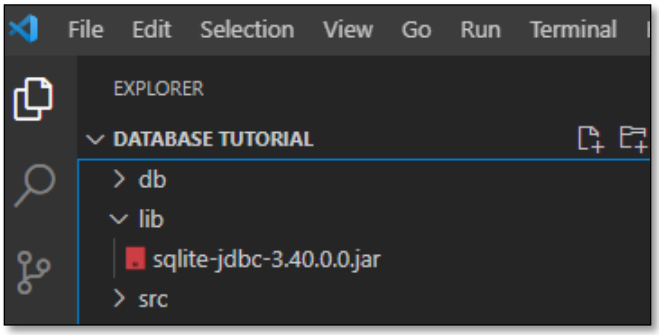
1. Create a folder for your database project.
2. Right Click your project folder → **Open with Code.**



3. Create a **lib, src,** and **db** folder as shown.



4. Download the latest JDBC sqlite jar file from https://github.com/xerial/sqlite-jdbc/releases
5. Copy it into the lib folder as shown.



Tutorial 2: DBController

Create the following file in the src folder. This will contain all the code for controlling the database. Let's create our database file.

DBController.java

```
1   /*
2    * Filename: DBController.java
3    * Written by:
4    * Written on:
5    * Connect to or create an SQLite database with Java
6    */
7
8   import java.sql.Connection;
9   import java.sql.DriverManager;
10  import java.sql.SQLException;
11  import java.sql.Statement;
12  import java.sql.PreparedStatement;
13  import java.sql.ResultSet;
14
15  public class DBController {
```

```java
    // ------------------- CREATE TABLES ---------------------------------//
    public void createTables() {
        try {
            // Connection string from jdbc to database file
            String url = "jdbc:sqlite:./db/data.db";
            // Create connection to database:
            // Create database if it doesn't exist, attach if it does
            Connection conn = DriverManager.getConnection(url);
            System.out.println("Database connected.");

            // Create statement object that uses
            // the connection to execute SQL statements
            Statement statement = conn.createStatement();

            // ------------------ DROP TABLE ---------------------------- //
            // Drop tables for testing purposes
            statement.execute("DROP TABLE IF EXISTS tbl_sale");
            statement.execute("DROP TABLE IF EXISTS tbl_product");
            statement.execute("DROP TABLE IF EXISTS tbl_customer");

            String CreateCustomerSQL = """
                        CREATE TABLE IF NOT EXISTS tbl_customer(
                        cust_id INTEGER PRIMARY KEY,
                        cust_fname TEXT,
                        cust_lname TEXT
                    );""";
            statement.execute(CreateCustomerSQL);
```

```
202
203             String CreateProductSQL = """
204                     CREATE TABLE IF NOT EXISTS tbl_product
205                     (
206                         prod_id INTEGER PRIMARY KEY,
207                         prod_name TEXT,
208                         prod_price REAL
209                     );""";
210             statement.execute(CreateProductSQL);
211
212             String CreateSaleSQL = """
213                     CREATE TABLE IF NOT EXISTS tbl_sale
214                     (
215                         sale_id INTEGER PRIMARY KEY,
216                         cust_id INTEGER,
217                         prod_id INTEGER,
218                         sale_quantity INTEGER,
219                         FOREIGN KEY (cust_id) REFERENCES tbl_customer(cust_id)
220                         FOREIGN KEY (prod_id) REFERENCES tbl_product(prod_id)
221                     );""";
222             statement.execute(CreateSaleSQL);
223
224             System.out.println("Table created.");
225             // Close resources
226             statement.close();
227             conn.close();
228         } catch (SQLException e) {
229             System.out.println("Something went wrong: " + e.getMessage());
230             e.printStackTrace();
231         }
232     }
233 }
```

Add a file named **POS1CreateTable.java** to the src folder.

```java
/*
 * Filename: POS1CreateTable.java
 * Written by:
 * Written on:
 * Connect to or create an SQLite database with Java
 */

public class POS1CreateTable {
    Run | Debug
    public static void main(String[] args) {
        DBController dbOperations = new DBController();

        dbOperations.createTables();
    }
}
```

Example run:

```
Database connected.
Table created.
```

Tutorial 3: Insert Customer Record

Add the following to **DBController.java**

```java
    // ------------------ INSERT CUSTOMER --------------------------------/
    public void insertCustomer(
            int custID,
            String custFname,
            String custLname) {
        try {
            // Connection string from jdbc to database file
            String url = "jdbc:sqlite:./db/data.db";
            // Create connection to database:
            // Create database if it doesn't exist, attach if it does
            Connection conn = DriverManager.getConnection(url);
            // To avoid SQL injection attacks, the data is set
            // into the preparedStatement rather than being in the SQL code
            String SQLInsert = """
                    INSERT INTO tbl_customer
                    (cust_id, cust_fname, cust_lname) VALUES(?, ?, ?);
                    """;
            PreparedStatement ps = conn.prepareStatement(SQLInsert);
            // To avoid SQL injection attacks, the data is set
            // into the preparedStatement rather than being in the SQL code
            ps.setInt(1, custID);
            ps.setString(2, custFname);
            ps.setString(3, custLname);
            ps.executeUpdate();

            System.out.println("Customer record inserted.");
            // Close resources
            ps.close();
            conn.close();
        } catch (SQLException e) {
            System.out.println("Something went wrong: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

POS2InsertCustomer.java

```java
/*
* Filename: POS2InsertCustomer.java
* Written by:
* Written on:
* Connect to or create an SQLite database with Java
*/

public class POS2InsertCustomer {
    Run | Debug
    public static void main(String[] args) {

        DBController dbOperations = new DBController();

        dbOperations.createTables();
        dbOperations.insertCustomer(0, "William", "Loring");
        dbOperations.insertCustomer(1, "Wyatt", "Earp");
    }
}
```

Example run:

```
Database connected.
Tables created.
Customer record inserted.
Customer record inserted.
```

Tutorial 4: Insert Product

Add the following to **DBController.java**

```java
    // ---------------------- INSERT PRODUCT -----------------------------//
    public void insertProduct(
            int prodID,
            String prodName,
            Double prodPrice) {
        try {
            // Connection string from jdbc to database file
            String url = "jdbc:sqlite:./db/data.db";
            // Create connection to database:
            // Create database if it doesn't exist, attach if it does
            Connection conn = DriverManager.getConnection(url);
            // The data is set into the preparedStatement
            // rather than being in the SQL code.
            String SQLInsert = """
                    INSERT INTO tbl_product
                    (prod_id, prod_name, prod_price) VALUES(?, ?, ?);
                    """;
            PreparedStatement ps = conn.prepareStatement(SQLInsert);
            // The data is set into the preparedStatement
            // rather than being in the SQL code.
            ps.setInt(1, prodID);
            ps.setString(2, prodName);
            ps.setDouble(3, prodPrice);
            ps.executeUpdate();

            System.out.println("Product record inserted.");
            // Close resources
            ps.close();
            conn.close();
        } catch (SQLException e) {
            System.out.println("Something went wrong: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

POS3InsertProduct.java

```java
/*
* Filename: POS3InsertProduct.java
* Written by:
* Written on:
* Connect to or create an SQLite database with Java
*/

public class POS3InsertProduct {
    Run | Debug
    public static void main(String[] args) {

        DBController dbOperations = new DBController();

        dbOperations.createTables();
        dbOperations.insertCustomer(0, "William", "Loring");
        dbOperations.insertCustomer(1, "Wyatt", "Earp");
        dbOperations.insertProduct(0, "Hammer", 10.95);
        dbOperations.insertProduct(1, "Saw", 15.95);
    }
}
```

Example run:

```
Database connected.
Tables created.
Customer record inserted.
Customer record inserted.
Product record inserted.
Product record inserted.
```

Tutorial 5: Insert Sale

Add the following to **DBController.java**

```java
    // ------------------ INSERT SALE ------------------------------------//
    public void insertSale(
            int saleID,
            int custID,
            int prodID,
            int saleQuantity) {
        try {
            // Connection string from jdbc to database file
            String url = "jdbc:sqlite:./db/data.db";
            // Create connection to database:
            // Create database if it doesn't exist, attach if it does
            Connection conn = DriverManager.getConnection(url);
            // The data is set into the preparedStatement
            // rather than being in the SQL code.
            String SQLInsert = """
                    INSERT INTO tbl_sale
                    (sale_id, cust_id, prod_id, sale_quantity)
                    VALUES(?, ?, ?, ? );
                    """;
            PreparedStatement ps = conn.prepareStatement(SQLInsert);
            // The data is set into the preparedStatement
            // rather than being in the SQL code.
            ps.setInt(1, prodID);
            ps.setInt(2, custID);
            ps.setInt(3, prodID);
            ps.setInt(4, saleQuantity);
            ps.executeUpdate();

            System.out.println("Sale record inserted.");
            // Close resources
            ps.close();
            conn.close();
        } catch (SQLException e) {
            System.out.println("Something went wrong: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

POS4InsertSale.java

```java
/*
* Filename: POS4InsertSale.java
* Written by:
* Written on:
* Connect to or create an SQLite database with Java
*/

public class POS4InsertSale {
    Run | Debug
    public static void main(String[] args) {

        DBController dbOperations = new DBController();

        dbOperations.createTables();
        dbOperations.insertCustomer(0, "William", "Loring");
        dbOperations.insertCustomer(1, "Wyatt", "Earp");
        dbOperations.insertProduct(0, "Hammer", 10.95);
        dbOperations.insertProduct(1, "Saw", 15.95);
        dbOperations.insertSale(0, 0, 0, 10);
        dbOperations.insertSale(1, 1, 1, 1);
        dbOperations.insertSale(2, 1, 0, 2);
    }
}
```

Tutorial 6: Fetch All Records

DBController.java

```java
    // ------------------ FETCH ALL RECORDS ------------------------------//
    public void fetchAllRecords() {
        try {
            // Connection string from jdbc to database file
            String url = "jdbc:sqlite:./db/data.db";
            // Create connection to database:
            // Create database if it doesn't exist, attach if it does
            Connection conn = DriverManager.getConnection(url);
            System.out.println("Database connected.");
            // Create statement object that uses
            // the connection to execute SQL statements
            Statement statement = conn.createStatement();
            // Fetch all records query to get all records
            // sorted by last name
            String 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;
                    """;
            statement.execute(SQL);
            // Get results from SQL query into results object
            ResultSet results = statement.getResultSet();
            // results.next() moves the cursor through
            // the results one record at a time
            while (results.next()) {
                System.out.println(
                        results.getString("cust_fname") +
                                " " + results.getString("cust_lname") +
                                " \n" + results.getString("sale_quantity") +
                                " " + results.getString("prod_name"));
            }
            // Close resources
            statement.close();
            conn.close();
        } catch (SQLException e) {
            System.out.println("Something went wrong: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

POS5FetchAllRecords.java

```java
/*
* Filename: POS5FetchAllRecords.java
* Written by:
* Written on:
* Connect to or create an SQLite database with Java
*/

public class POS5FetchAllRecords {
    Run | Debug
    public static void main(String[] args) {

        DBController dbOperations = new DBController();

        dbOperations.createTables();
        dbOperations.insertCustomer(0, "William", "Loring");
        dbOperations.insertCustomer(1, "Wyatt", "Earp");
        dbOperations.insertProduct(0, "Hammer", 10.95);
        dbOperations.insertProduct(1, "Saw", 15.95);
        dbOperations.insertSale(0, 0, 0, 10);
        dbOperations.insertSale(1, 1, 1, 1);
        dbOperations.insertSale(2, 1, 0, 2);
        dbOperations.fetchAllRecords();
    }
}

```

Example run:

```
Tables created.
Customer record inserted.
Customer record inserted.
Product record inserted.
Product record inserted.
Sale record inserted.
Sale record inserted.
Sale record inserted.
Database connected.
Wyatt Earp
1 Saw
Wyatt Earp
2 Hammer
William Loring
10 Hammer
```

Assignment Submission

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