Java Chapter 12 Database SQLite

Contents

Time required: 90 minutes

SQLite and Java

Databases offer numerous functionalities by which one can manage large amounts of information easily over the web and high-volume data input and output over a typical file such as a text file. SQL is a query language and is very popular in databases. Many websites use MySQL. SQLite is a "light" version that works over syntax very similar to SQL.

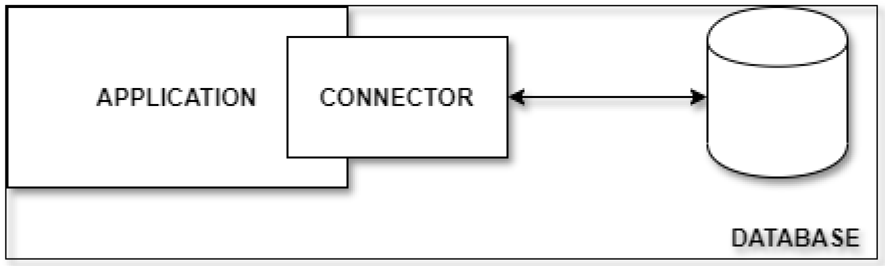
SQLite is a self-contained, high-reliability, embedded, full-featured, public-domain, SQL database engine. It is the most used database engine on the world wide web.

SQLite has the following features.

1. Serverless

2. Self-Contained
3. Zero-Configuration
4. Transactional
5. Single-Database file

SQLite does not require a server to run. The SQLite database is joined with the application that accesses the database. SQLite database read and write directly from the database files stored on disk and applications interact with that SQLite database.



Self-Contained

SQLite does not need any external dependencies like an operating system or external library. This feature of SQLite helps especially in embedded devices like iPhones, Android phones, game consoles, handheld media players, etc.

Zero-Configuration

Zero-configuration means no setup or administration needed. Because of the serverless architecture, you don’t need to “install” SQLite before using it. There is no server process that needs to be configured, started, and stopped.

Transactional

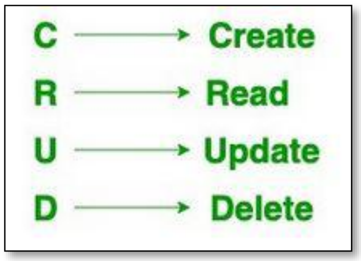
Transactional means they are atomic, consistent, isolated, and durable(ACID). All transactions in SQLite are fully ACID-compliant. In other words, all changes within a transaction take place completely or not at all even when an unexpected situation like application crash, power failure, or operating system crash occurs.

Single-Database

SQLite is a single database that means it allows a single database connection to access multiple database files simultaneously. These features bring many nice features like joining tables in different databases or copying data between databases in a single command. SQLite also uses dynamic types for tables. It means you can store any value in any column, regardless of the data type.

CRUD

A database provides four major operations, usually referred to as CRUD.



The corresponding SQL commands. Notice that SQL commands are typically in UPPER CASE. This is a convention; SQL does not require UPPER CASE. It makes the SQL commands easier to pick out of an SQL statement

CREATE: Create new tables and records

SELECT: Select all or selected fields and records

UPDATE: Modify existing tables and records

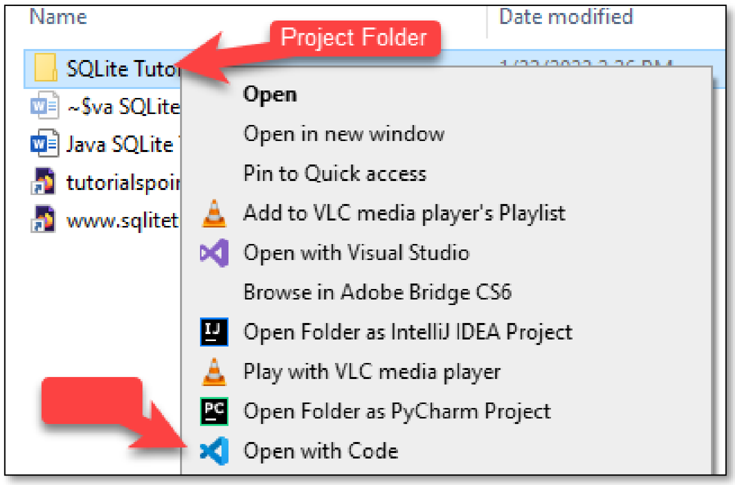
DELETE: Delete records

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

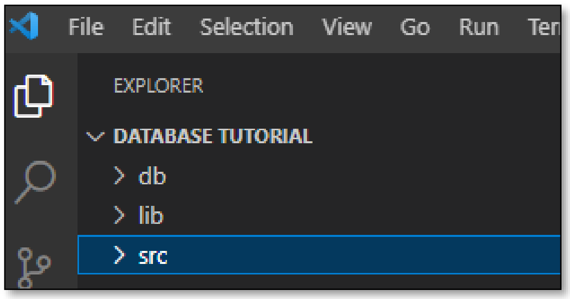
We are going to connect with our SQLite database using Java JDBC (Java Database Connectivity) API. We will be using a JDBC driver call the SQLiteJDBC Package. The SQLiteJDBC package contains both Java classes, as well as native SQLite libraries for Windows, Mac OS X, and Linux.

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.**

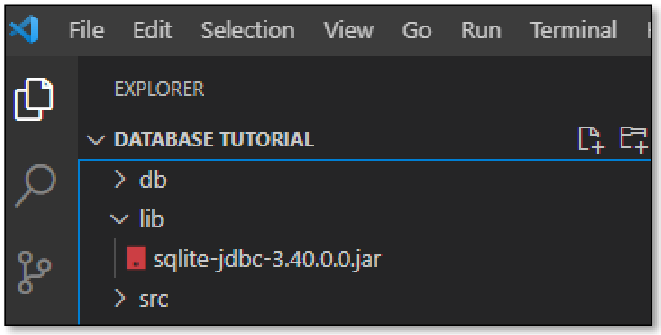


3. Create a **lib, src,** and **db** folder as shown.



4. Download the JDBC sqlite jar file from https://github.com/xerial/sqlite-jdbc/releases

5. Copy it into the lib folder as shown.



Tutorial 2: Connect to an SQLite Database

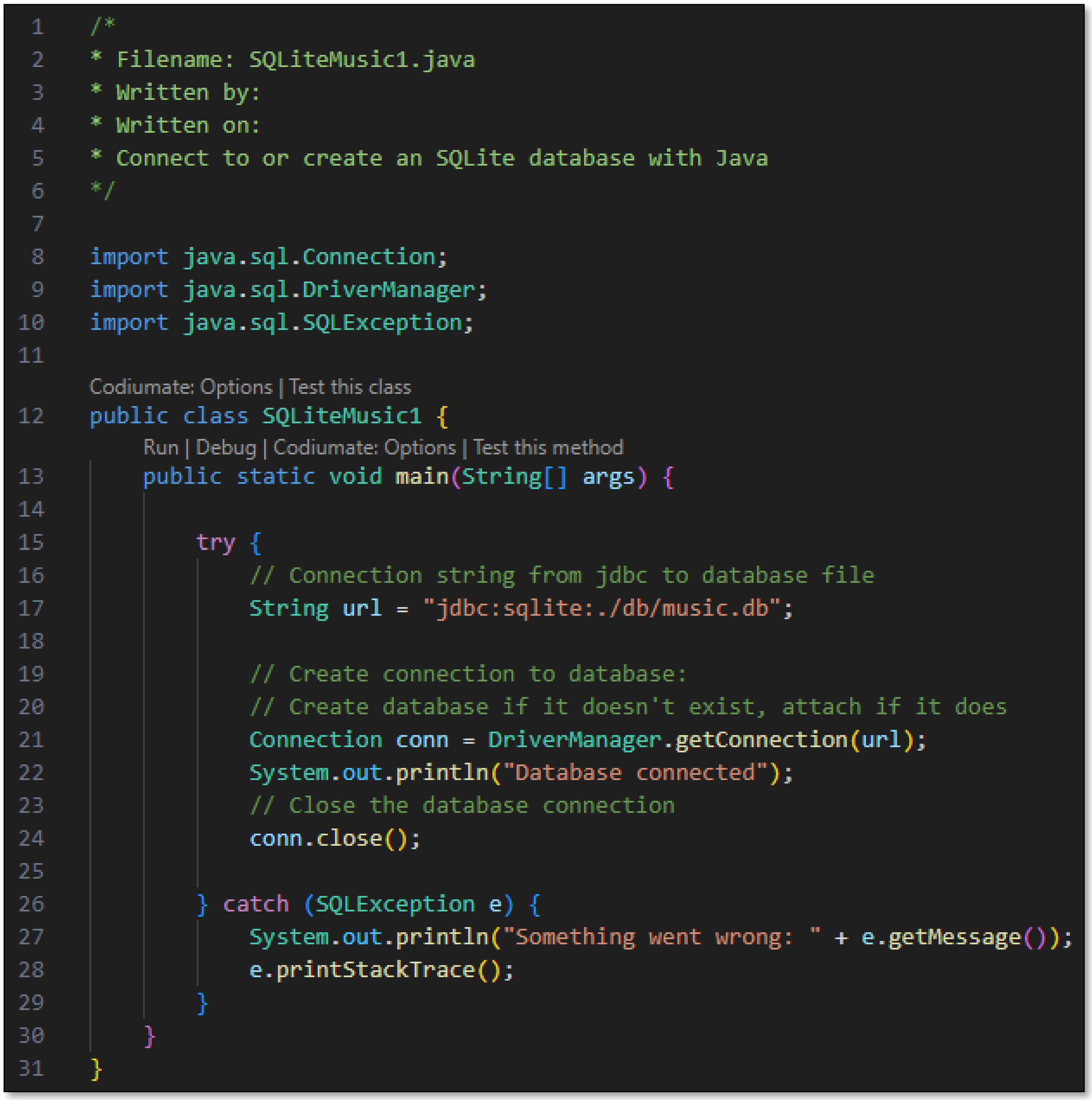
Please keep copies of each tutorial as you go in case something goes wrong. Use the name shown in the tutorial screenshot.

The SQLite JDBC driver allows you to load an SQLite database from the file system using the following connection string:

```
jdbc:sqlite:sqlite_database_file_name
```

1. In your project folder under src → create a Java file named: **SQLiteMusic.java**

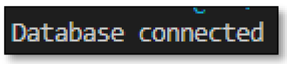
2. Insert the following code to create and or connect to a database.


Codiumate: Options | Test this class
Run | Debug | Codiumate: Options | Test this method

```java
/*
* Filename: SQLiteMusic1.java
* Written by:
* Written on:
* Connect to or create an SQLite database with Java
*/

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class SQLiteMusic1 {
    public static void main(String[] args) {

        try {
            // Connection string from jdbc to database file
            String url = "jdbc:sqlite:./db/music.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");
            // Close the database connection
            conn.close();

        } catch (SQLException e) {
            System.out.println("Something went wrong: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

3. You should see **music.db** in your db folder.

Example run:



```
Database connected
```

Creating Tables

In an SQL database, data is stored in tables. Tables define a set of columns and fields, much like a spreadsheet. They contain 0 or more rows with data for each of the defined fields.

Let's create a table named track that tracks the following data for some music tracks.

track_id	**track_name**	**track_time**
0	Stairway to Heaven	5.25
1	Old Time Rock and Roll	3.25
2	Sister Christian	2.25
3	Ramblin Man	1.25

The track table will track a value for id, name, and time for each track. The id is the primary key for the table. A primary key is a special relational database table column (or combination of columns) designated to uniquely identify each table record.

We can create a **track** table in SQL:

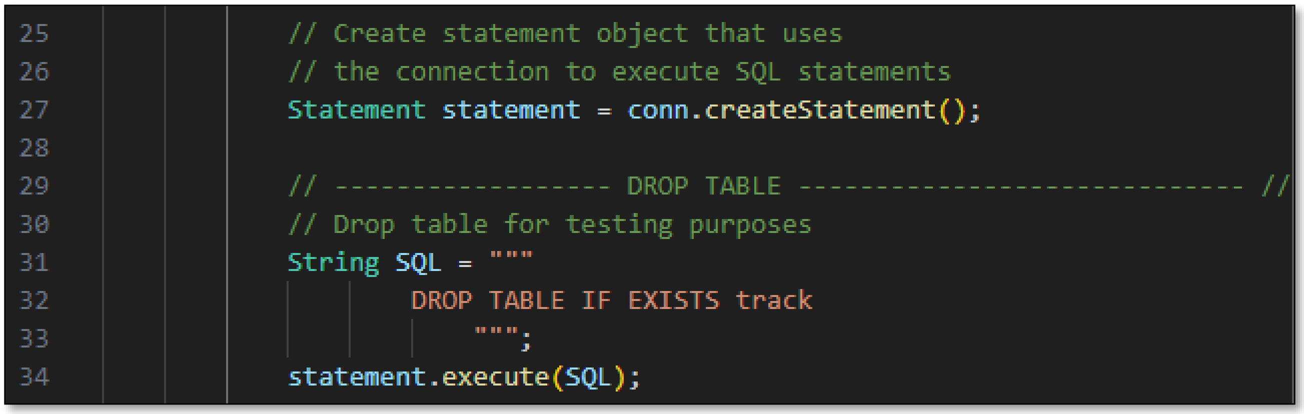
```
CREATE TABLE IF NOT EXISTS track " +
                    "(track_id INTEGER PRIMARY KEY, " +
                    "track_name TEXT, " +
                    "track_time REAL)")
```

- If the table does not exist, we create it. If it does exist, we skip this step.
- The "CREATE TABLE ..." string is a SQL statement that creates a table named track with the columns described earlier:
    - **id** of type integer PRIMARY KEY
    - **name** of type text
    - **time** of type real
- A primary key is a unique value that provides a unique identifier for each record. A primary key is typically a sequence of whole numbers (integer).

Tutorial 3: Create a Table

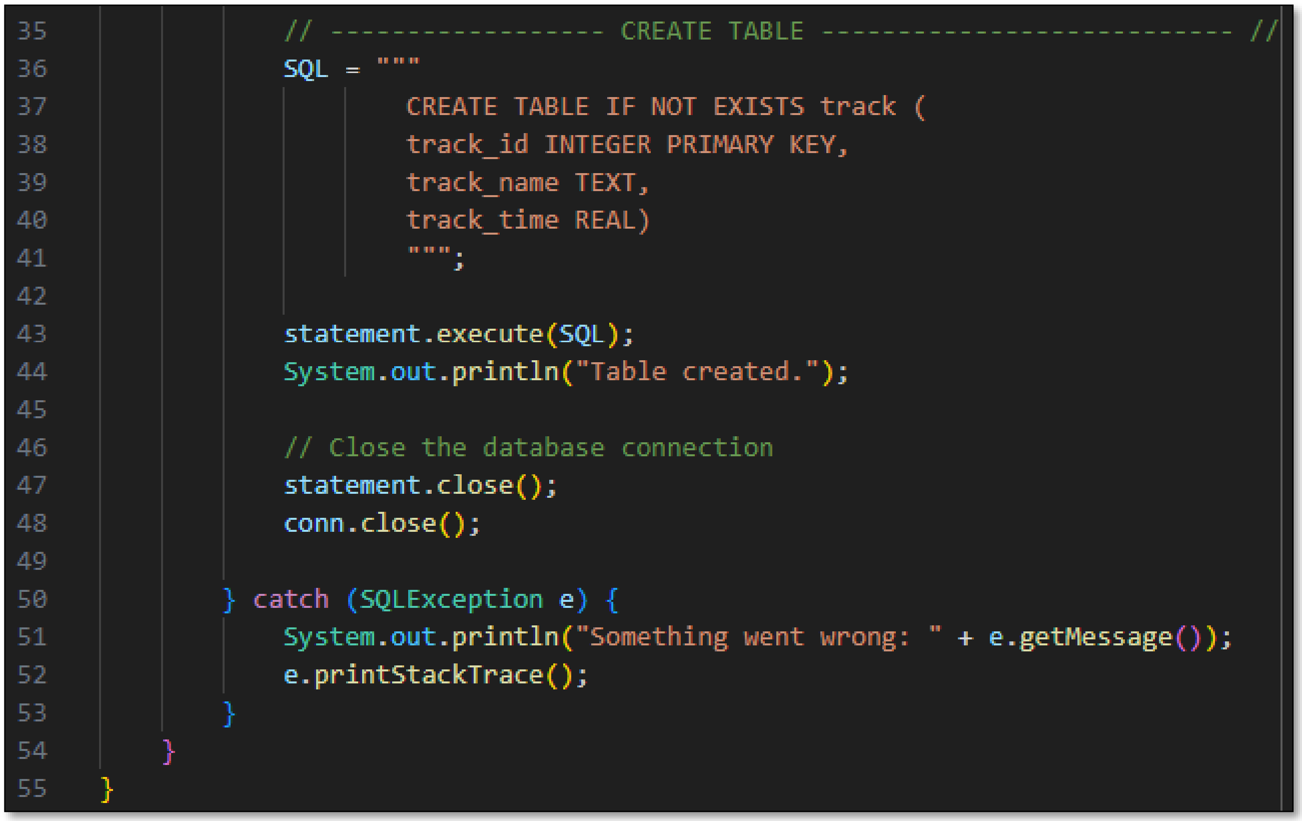
At the top of the file add: **import java.sql.Statement;**

Add the following code after the database connection.



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

            // ------------------ DROP TABLE ---------------------------- //
            // Drop table for testing purposes
            String SQL = """
                    DROP TABLE IF EXISTS track
                        """;
            statement.execute(SQL);
```

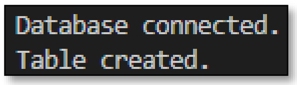
Create a Statement object. This is much like the cursor in Python SQLite. Execute the SQL on the database



```java
            // ------------------ CREATE TABLE --------------------------- //
            SQL = """
                    CREATE TABLE IF NOT EXISTS track (
                    track_id INTEGER PRIMARY KEY,
                    track_name TEXT,
                    track_time REAL)
                    """;

            statement.execute(SQL);
            System.out.println("Table created.");

            // Close the database connection
            statement.close();
            conn.close();

        } catch (SQLException e) {
            System.out.println("Something went wrong: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

Create the track table using a statement.

Example run:



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

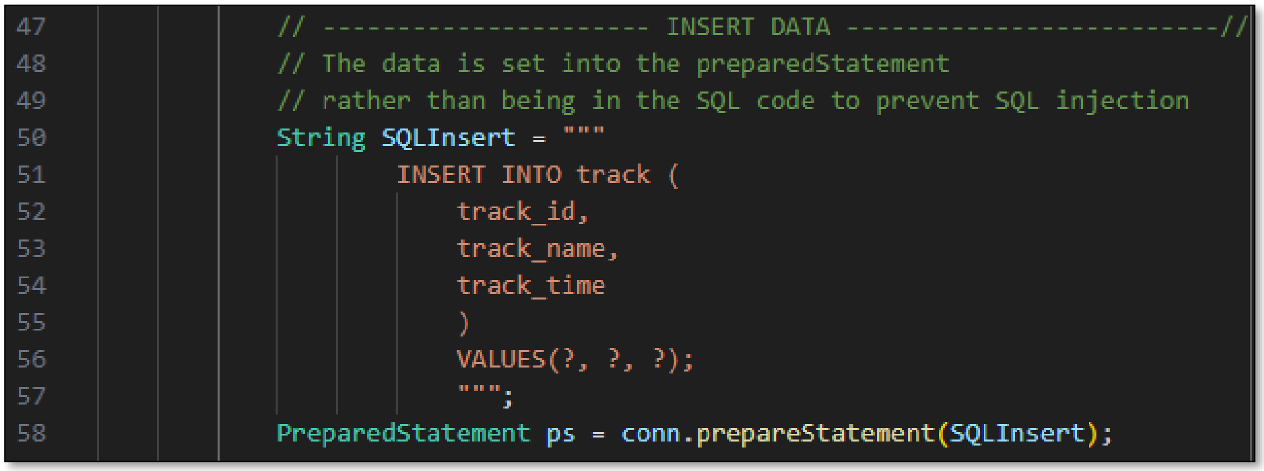
Tutorial 4: Insert Records

At the top of the file add: **import java.sql.PreparedStatement;**

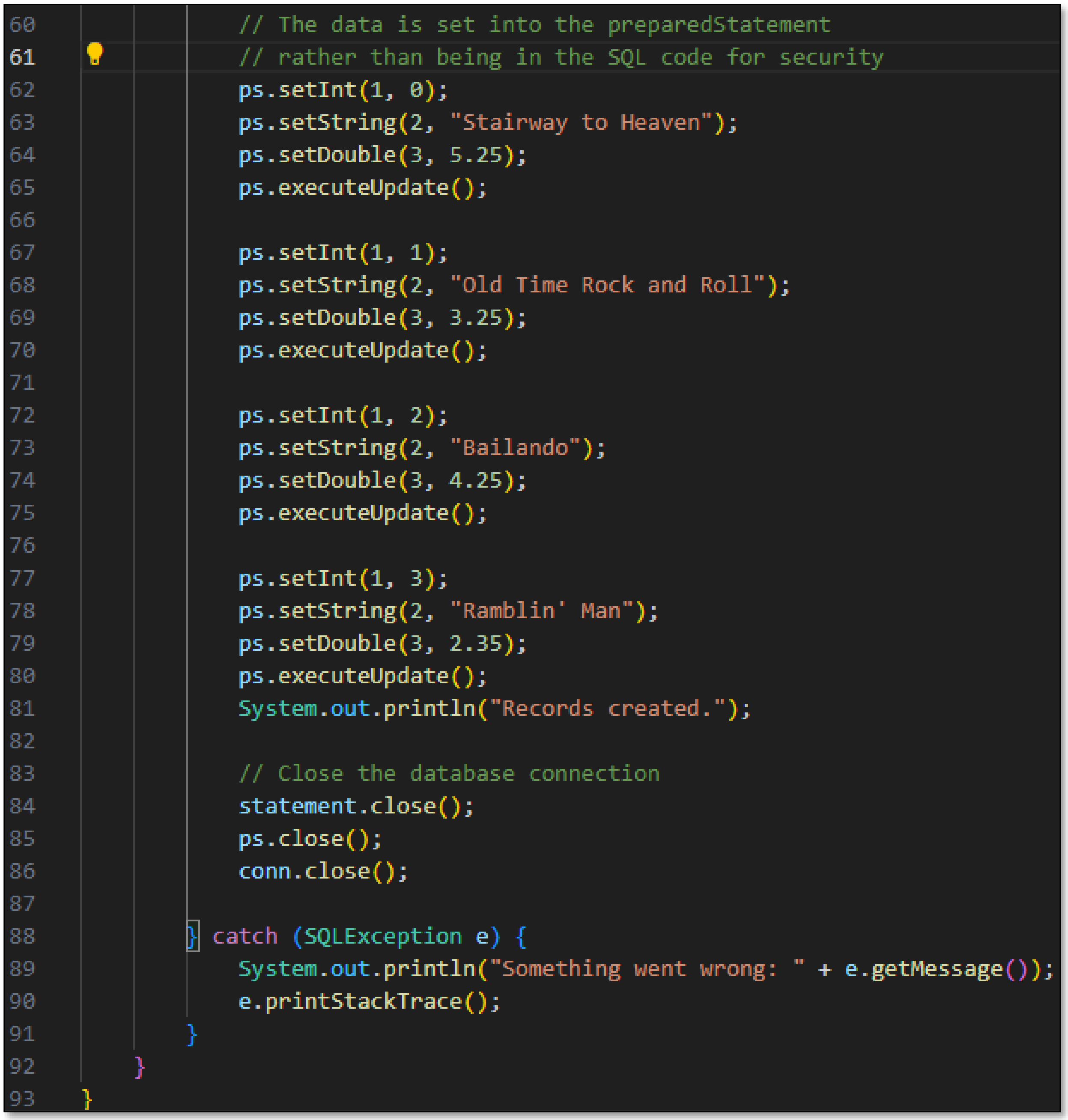
We have a created a table. This is SQL for inserting records.

```
"INSERT INTO track VALUES (0, 'Stairway to Heaven', 5.25)"
```

- INSERT INTO track specifies that we are inserting into the track table.
- VALUES says that the values are coming up next.
- The values must match the datatypes of the table fields.

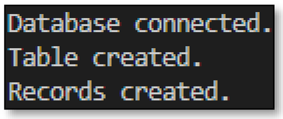


```java
            // --------------------- INSERT DATA ------------------------//
            // The data is set into the preparedStatement
            // rather than being in the SQL code to prevent SQL injection
            String SQLInsert = """
                    INSERT INTO track (
                        track_id,
                        track_name,
                        track_time
                        )
                        VALUES(?, ?, ?);
                        """;
            PreparedStatement ps = conn.prepareStatement(SQLInsert);
```



```java
            // The data is set into the preparedStatement
            // rather than being in the SQL code for security
            ps.setInt(1, 0);
            ps.setString(2, "Stairway to Heaven");
            ps.setDouble(3, 5.25);
            ps.executeUpdate();

            ps.setInt(1, 1);
            ps.setString(2, "Old Time Rock and Roll");
            ps.setDouble(3, 3.25);
            ps.executeUpdate();

            ps.setInt(1, 2);
            ps.setString(2, "Bailando");
            ps.setDouble(3, 4.25);
            ps.executeUpdate();

            ps.setInt(1, 3);
            ps.setString(2, "Ramblin' Man");
            ps.setDouble(3, 2.35);
            ps.executeUpdate();
            System.out.println("Records created.");

            // Close the database connection
            statement.close();
            ps.close();
            conn.close();

        } catch (SQLException e) {
            System.out.println("Something went wrong: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

Example run:

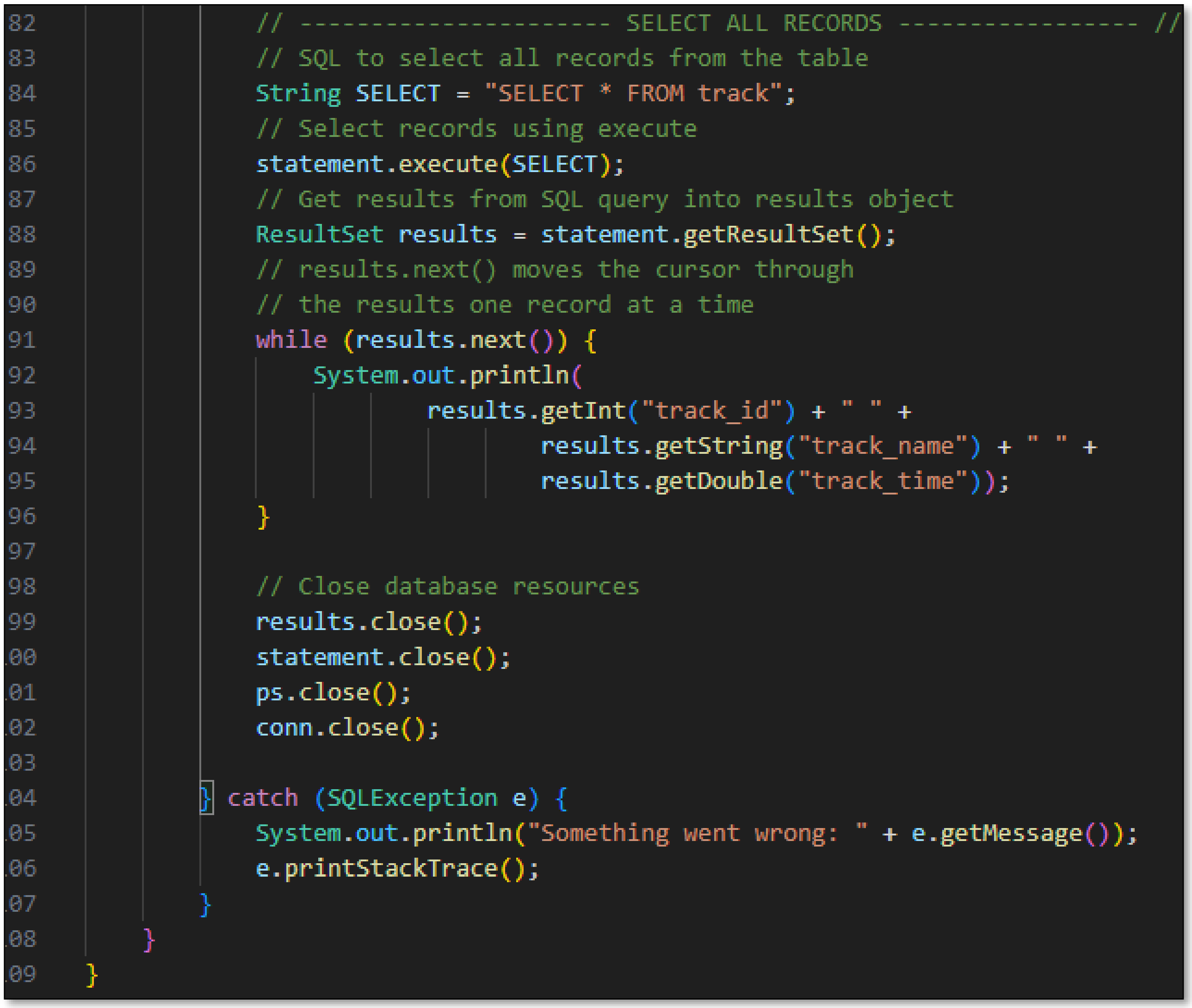


```
Database connected.
Table created.
Records created.
```

Tutorial 5: Selecting Data

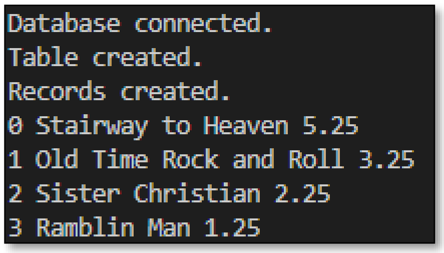
Selecting and then fetching the data from records is simple as inserting them. The execute method uses the SQL command of getting all the data from the table using **"SELECT * from table_name"**. The results are stored in a **ResultSet** object which we access with a loop.

At the top of the file add: **import java.sql.ResultSet;**



```java
            // ---------------------- SELECT ALL RECORDS ---------------- //
            // SQL to select all records from the table
            String SELECT = "SELECT * FROM track";
            // Select records using execute
            statement.execute(SELECT);
            // 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.getInt("track_id") + " " +
                                results.getString("track_name") + " " +
                                results.getDouble("track_time"));
            }

            // Close database resources
            results.close();
            statement.close();
            ps.close();
            conn.close();

        } catch (SQLException e) {
            System.out.println("Something went wrong: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

Example run:


```
Database connected.
Table created.
Records created.
0 Stairway to Heaven 5.25
1 Old Time Rock and Roll 3.25
2 Sister Christian 2.25
3 Ramblin Man 1.25
```

Updating Data

Rows in an SQLite database can be modified using UPDATE SQL statement. We can update single columns as well as multiple columns.

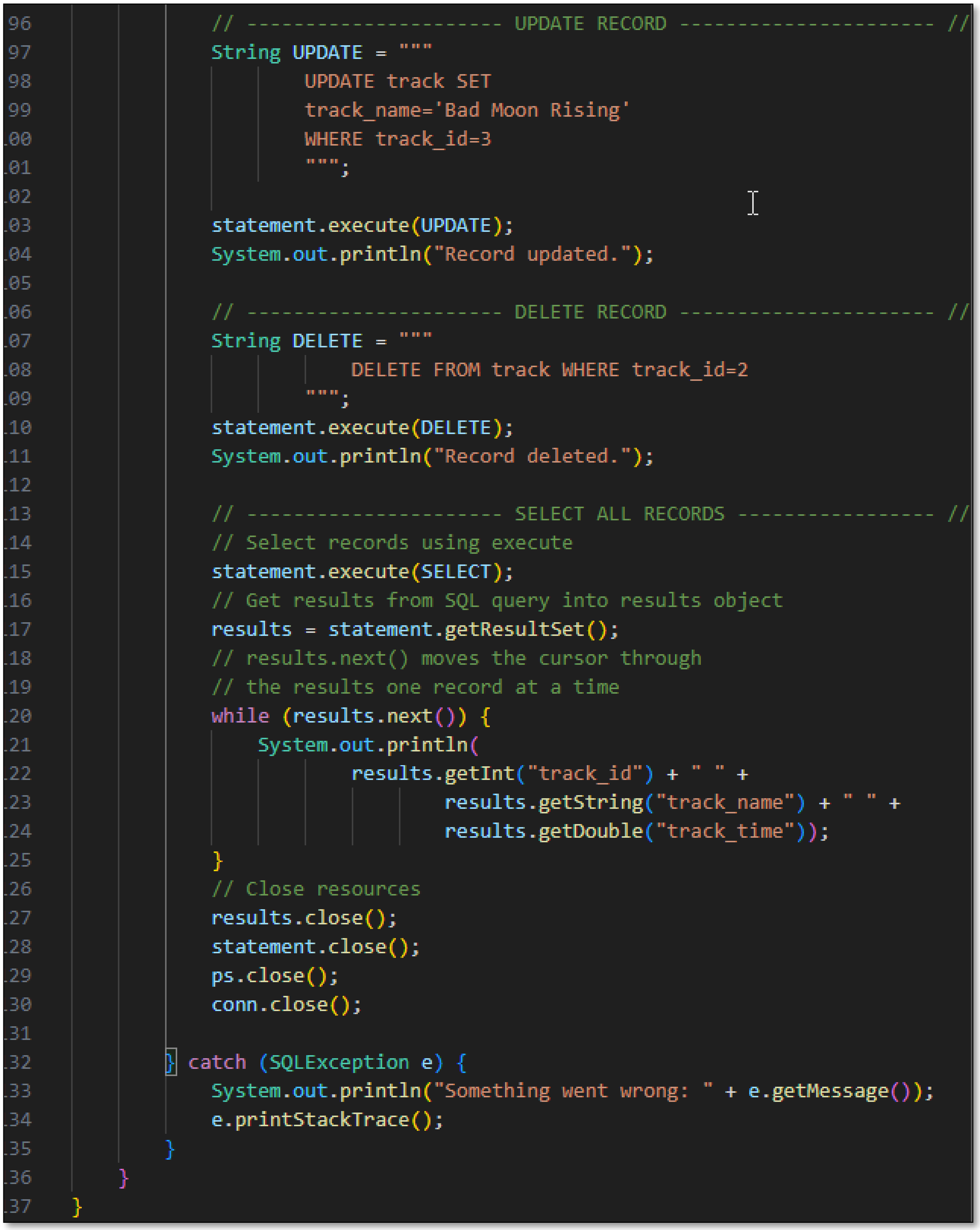
This is the general form to update existing table data.

```
UPDATE table_name SET column1 = value1, column2 = value2 WHERE condition;
```

In the above syntax, the SET statement is used to set new values to the particular column, and the WHERE clause is used to select the rows for which the columns are needed to be updated.

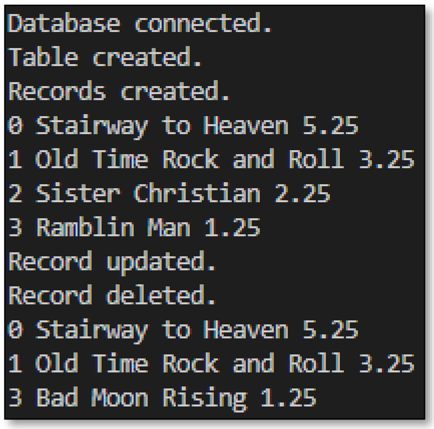
Tutorial 6: Updating and Deleting Data

Add the following code to update and delete a record.



```
            // ---------------------- UPDATE RECORD ---------------------- //
            String UPDATE = """
                    UPDATE track SET
                    track_name='Bad Moon Rising'
                    WHERE track_id=3
                    """;

            statement.execute(UPDATE);
            System.out.println("Record updated.");

            // ---------------------- DELETE RECORD ---------------------- //
            String DELETE = """
                        DELETE FROM track WHERE track_id=2
                    """;
            statement.execute(DELETE);
            System.out.println("Record deleted.");

            // ---------------------- SELECT ALL RECORDS ----------------- //
            // Select records using execute
            statement.execute(SELECT);
            // Get results from SQL query into results object
            results = statement.getResultSet();
            // results.next() moves the cursor through
            // the results one record at a time
            while (results.next()) {
                System.out.println(
                        results.getInt("track_id") + " " +
                                results.getString("track_name") + " " +
                                results.getDouble("track_time"));
            }
            // Close resources
            results.close();
            statement.close();
            ps.close();
            conn.close();

        } catch (SQLException e) {
            System.out.println("Something went wrong: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

Example run:



```
Database connected.
Table created.
Records created.
0 Stairway to Heaven 5.25
1 Old Time Rock and Roll 3.25
2 Sister Christian 2.25
3 Ramblin Man 1.25
Record updated.
Record deleted.
0 Stairway to Heaven 5.25
1 Old Time Rock and Roll 3.25
3 Bad Moon Rising 1.25
```

Assignment Submission

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