SQLite JAVA Tutorial

In this tutorial we will learn how to use JAVA to connect SQLite database and CREATE Tables, then and see how to use the INSERT, UPDATE, DELETE and SELECT operations on SQLite tables with examples.

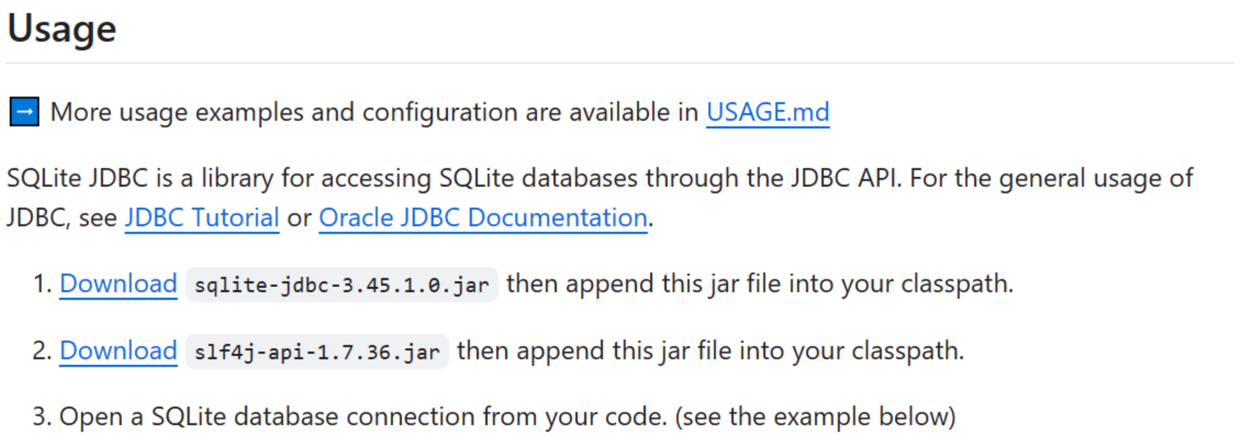
SQLite-JAVA Interface

We can interact with SQLite in JAVA language using the SQLite-JDBC driver. The JDBC driver is a Java package which contains both JAVA classes and SQLite libraries to perform different operations like connect to the database, create tables, insert data in tables, etc.

Before we proceed to interact with SQLite using JAVA language, first we need to make sure that the SQLite-JDBC driver (sqlite-jdbc-version#.jar) is downloaded and installed. To do this, you can search Google for the driver or go to the Xerial github site as follows:

1- visit the github site:
https://github.com/xerial/sqlite-jdbc

2- Go to the download section:



Usage

➡ More usage examples and configuration are available in USAGE.md

SQLite JDBC is a library for accessing SQLite databases through the JDBC API. For the general usage of JDBC, see JDBC Tutorial or Oracle JDBC Documentation.

1. Download `sqlite-jdbc-3.45.1.0.jar` then append this jar file into your classpath.
2. Download `slf4j-api-1.7.36.jar` then append this jar file into your classpath.
3. Open a SQLite database connection from your code. (see the example below)

3- Download both jar files:
`sqlite-jdbc-3.46.0.0.jar`
slf4j-api-1.7.36.jar

to an appropriate folder or your project folder (e.g D:\JDBC>) then append this folder's path to the classpath or put them in the same folder of your project.

Connect to SQLite Database using Java

Now we will connect to the SQLite database using JAVA JDBC library.
Notice:

- When you try to connect to a database you must put the connection statement in a try-catch block. If the database exists and connection is successful you will get a link to the database that will be used on all operations
- If the database does not exist, a new database will be created and connected to the program.
- Otherwise, an error message will be created.

The following JAVA program shows how to connect a database if it exists otherwise first it will create a database and then connect to it.

```
import java.sql.*;
public class DBUsingJava {

	public static void main( String args[] )
	{
		Connection c = null;
		try {
			Class.forName("org.sqlite.JDBC");
			c = 
		DriverManager.getConnection("jdbc:sqlite:SqliteJavaDB.db");

		} catch ( Exception e ) {
			System.err.println( e.getClass().getName() + ": " + 
		e.getMessage() );
		System.exit(0);
		}
		System.out.println("database successfully created");
	}
}
```

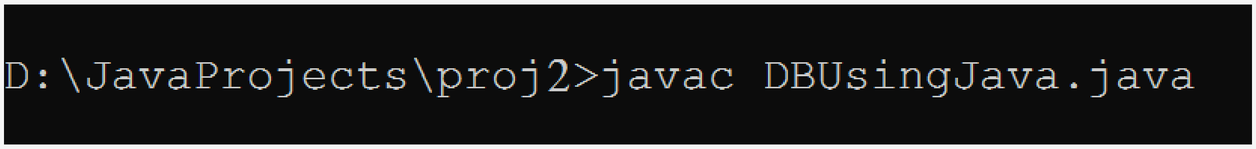
In this program we are trying to connect "**SqliteJavaDB.db**" if exists otherwise the program will create a new database in current folder. We assume that **sqlite-jdbc-3.45.1.0.jar** (**3.45.1.0 is the version number of**

the driver, yours might be different) is available at the same location where our program exists.

How to run this program:

1- **Compile the code**

```
In this case the code is stored in the folder d:\JavaProjects\Proj2>
```



```
D:\JavaProjects\proj2>javac DBUsingJava.java
```

2- Run the Program code:
Use the following command as is in one line:

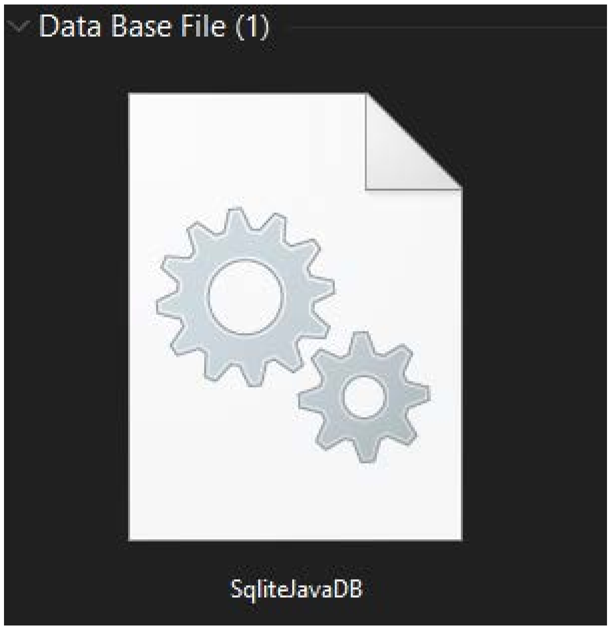
```
D:\JavaProjects\Proj2>
javac -cp ".;sqlite-jdbc-3.46.0.0.jar;slf4j-api-1.7.36.jar" DBUsingJava
```

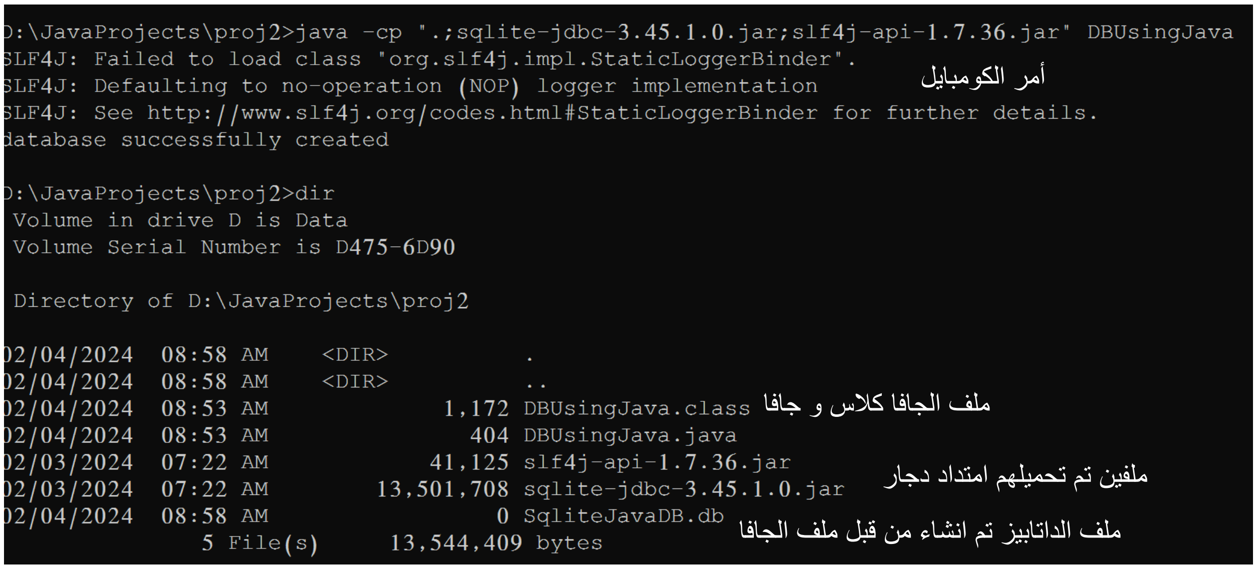
اسم الملف   إصدارات الملفات 3.46.0.0 و 1.7.36

إذا أعطيت لملفك اسم مختلف او كانت الملفات ذات اصدار مختلف وجب تغيير أمر الكومبايل

```
You will get the message:
"database successfully created successfully"
and a file called SqliteJavaDB will be created in your designate folder
```

and the database (SQLiteJavaDB.db) will be created as follows:





```
D:\JavaProjects\proj2>java -cp ".;sqlite-jdbc-3.45.1.0.jar;slf4j-api-1.7.36.jar" DBUsingJava
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
database successfully created

D:\JavaProjects\proj2>dir
 Volume in drive D is Data
 Volume Serial Number is D475-6D90

 Directory of D:\JavaProjects\proj2

02/04/2024  08:58 AM    <DIR>          .
02/04/2024  08:58 AM    <DIR>          ..
02/04/2024  08:53 AM             1,172 DBUsingJava.class
02/04/2024  08:53 AM               404 DBUsingJava.java
02/03/2024  07:22 AM            41,125 slf4j-api-1.7.36.jar
02/03/2024  07:22 AM        13,501,708 sqlite-jdbc-3.45.1.0.jar
02/04/2024  08:58 AM                 0 SqliteJavaDB.db
               5 File(s)     13,544,409 bytes
```

أمر الكومبايل
ملف الجافا كلاس و جافا
ملفين تم تحميلهم امتداد دجار
ملف الداتابيز تم انشاء من قبل ملف الجافا

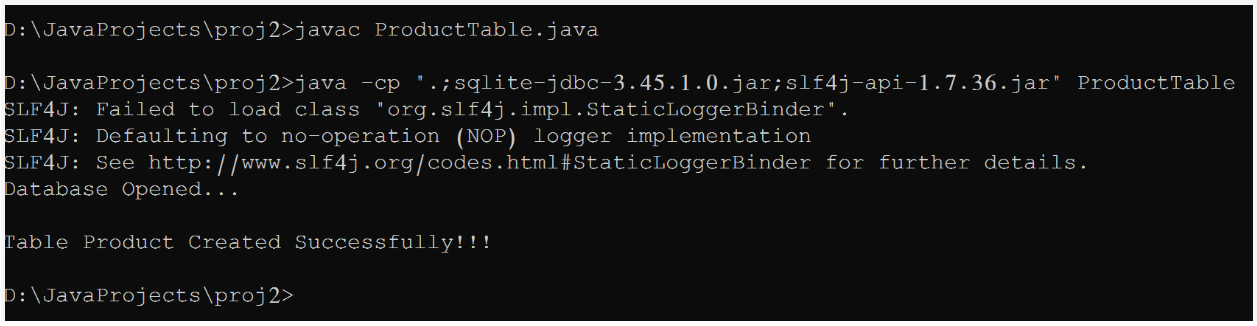
Create Table in SQLite Database using Java

Now, we will create a new table in newly created database **SqliteJavaDB.db** using java. The following code will do that:

```
import java.sql.*;
public class ProductTable
{
      public static void main( String args[] )
      {
            Connection c = null;
            Statement stmt = null;
      try {
      Class.forName("org.sqlite.JDBC");
      c = DriverManager.getConnection("jdbc:sqlite:SqliteJavaDB.db");
      System.out.println("Database Opened...\n");
      stmt = c.createStatement();
      String sql = "CREATE TABLE Product " +
      "(p_id INTEGER PRIMARY KEY AUTOINCREMENT," +
      " p_name TEXT NOT NULL, " +
      " price REAL NOT NULL, " +
      " quantity INTEGER) " ;
      stmt.executeUpdate(sql);
      stmt.close();
      c.close();
      } catch ( Exception e ) {
      System.err.println( e.getClass().getName() + ": " + e.getMessage()
      );
      System.exit(0);
      }
      System.out.println("Table Product Created Successfully!!!");
      }
}
```

Now compile and run the program using the following commands.

```
D:\myCode>javac ProductTable.java
D:\myCode>java -cp ".;sqlite-jdbc-3.45.1.0.jar; slf4j-api-1.7.36.jar" ProductTable

Database Opened...

Table Product Created Successfully!!!
```



```
D:\JavaProjects\proj2>javac ProductTable.java

D:\JavaProjects\proj2>java -cp ".;sqlite-jdbc-3.45.1.0.jar;slf4j-api-1.7.36.jar" ProductTable
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
Database Opened...

Table Product Created Successfully!!!

D:\JavaProjects\proj2>
```

DML Operations Using Java

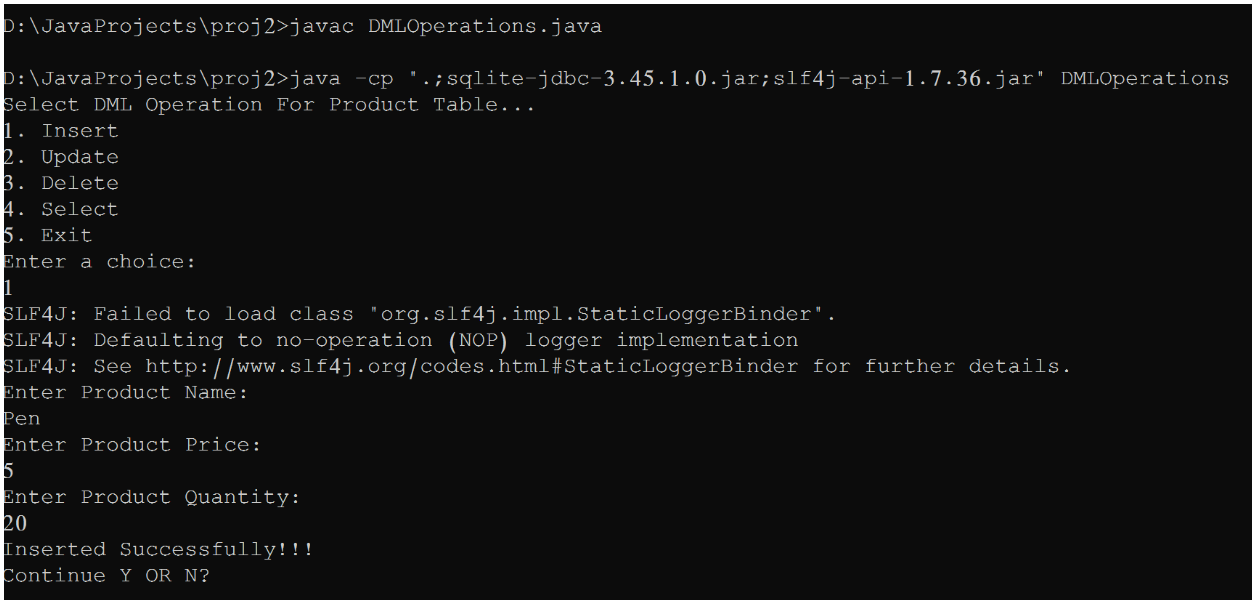
Now, we move to the Data Manipulation Language (DML) operations (insert, select, update, delete). These are the main operations that we can perform on a previously created table named Product using JAVA language.

Following program contains all 4 DML operations like INSERT, UPDATE, DELETE and SELECT.

```
import java.util.Scanner;
import java.sql.*;
public class DMLOperation
{
public static void main( String args[] )
{
String flag="Y";
do{
System.out.println("Select DML Operation For Product Table...");
System.out.println("1. Insert");
System.out.println("2. Update");
System.out.println("3. Delete");
System.out.println("4. Select");
System.out.println("5. Exit");
Scanner reader = new Scanner(System.in);
System.out.println("Enter a choice: ");
int n = reader.nextInt();
Connection c = null;
Statement stmt = null;
try {
Class.forName("org.sqlite.JDBC");
```

```java
c = DriverManager.getConnection("jdbc:sqlite:SqliteJavaDB.db");
c.setAutoCommit(false);
stmt = c.createStatement();
String name="",sql="";
float price=0.0f;
int quantity=0;
int id;
Scanner scanName;
switch(n){

case 1:
scanName=new Scanner(System.in);
System.out.println("Enter Product Name:");
name=scanName.nextLine();
System.out.println("Enter Product Price:");
price=scanName.nextFloat();
System.out.println("Enter Product Quantity:");
quantity=scanName.nextInt();
sql = "INSERT INTO Product (p_name,price,quantity) " +
"VALUES ('" +name+ "'," +
price + "," + quantity + ")";
stmt.executeUpdate(sql);
System.out.println("Inserted Successfully!!!");
break;

case 2:
System.out.println("Enter Product id:");
scanName=new Scanner(System.in);
id=scanName.nextInt();
System.out.println("Enter Product Name:");
scanName=new Scanner(System.in);
name=scanName.nextLine();
System.out.println("Enter Product Price:");
price=scanName.nextFloat();
System.out.println("Enter Product Quantity:");
quantity=scanName.nextInt();

sql = "UPDATE Product SET p_name = '"+ name + "',price=" + price
+",quantity=" + quantity +
" WHERE p_id=" +id ;

stmt.executeUpdate(sql);
System.out.println("Updated Successfully!!!");
break;

case 3:
System.out.println("Enter Product id:");
scanName=new Scanner(System.in);
id=scanName.nextInt();
sql="DELETE FROM Product WHERE p_id=" + id+";";
stmt.executeUpdate(sql);
System.out.println("Deleted Successfully!!!");
break;

case 4:
ResultSet rs = stmt.executeQuery("SELECT * FROM Product;");
System.out.println("ID\t Name\t\t Price\t Qty ");
while ( rs.next() ) {
id = rs.getInt("p_id");
name = rs.getString("p_name");
quantity = rs.getInt("quantity");
price = rs.getFloat("price");
System.out.println(id+"\t "+name+" \t "+price+"\t "+quantity);
}
rs.close();
break;

case 5:
System.exit(0);
break;

default:
System.out.println("Oops!!! Wrong Choice...");
break;
}
stmt.close();
c.commit();
c.close();
}
catch ( Exception e )
{
System.err.println( e.getClass().getName() + ": " + e.getMessage() );
System.exit(0);
}

System.out.println("Continue Y OR N?");
reader=new Scanner(System.in);
flag=reader.nextLine();

}while(flag.equalsIgnoreCase("Y"));
System.exit(0);
}
}
```

If you observe above program we are performing INSERT, UPDATE, DELETE and SELECT operations on table called "**Product**". Now, let's compile and run the program to examine the output like as shown below.



```
D:\JavaProjects\proj2>javac DMLOperations.java

D:\JavaProjects\proj2>java -cp ".;sqlite-jdbc-3.45.1.0.jar;slf4j-api-1.7.36.jar" DMLOperations
Select DML Operation For Product Table...
1. Insert
2. Update
3. Delete
4. Select
5. Exit
Enter a choice:
1
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
Enter Product Name:
Pen
Enter Product Price:
5
Enter Product Quantity:
20
Inserted Successfully!!!
Continue Y OR N?
```

```
D:\myCode> javac DMLOperations.java
D:\myCode>java -cp ".;sqlite-jdbc-3.45.1.0.jar; slf4j-api-1.7.36.jar" DMLOperations

Select DML Operation For Product Table...
1. Insert
2. Update
3. Delete
4. Select
5. Exit

Enter a choice:
1

Enter Product Name:
Pencil
Enter Product Price:
5
Enter Product Quantity:
50

Inserted Successfully!!!

Continue Y OR N?
Y
```

```
Select DML Operation For Product Table...
1. Insert
2. Update
3. Delete
4. Select
5. Exit

Enter a choice:
4

ID Name Price Qty
---- ------ ----- ----
1 Pencil 5.0 50

Continue Y OR N?
Y

Select DML Operation For Product Table...
1. Insert
2. Update
3. Delete
4. Select
5. Exit

Enter a choice:
2

Enter Product id:
1
Enter Product Name:
Sharpner
Enter Product Price:
10
Enter Product Quantity:
90

Updated Successfully!!!

Continue Y OR N?
Y

Select DML Operation For Product Table...
1. Insert
2. Update
3. Delete
4. Select
5. Exit
```

```
Enter a choice:
4

ID Name Price Qty
---- -------- ----- ----
1 Sharpner 10.0 90

Continue Y OR N?
Y

Select DML Operation For Product Table...
1. Insert
2. Update
3. Delete
4. Select
5. Exit

Enter a choice:
1

Enter Product Name:
Scale
Enter Product Price:
5
Enter Product Quantity:
60

Inserted Successfully!!!

Continue Y OR N?
Y

Select DML Operation For Product Table...
1. Insert
2. Update
3. Delete
4. Select
5. Exit

Enter a choice:
4

ID Name Price Qty
---- -------- ---- ---
1 Sharpner 10.0 90
2 Scale 5.0 60

Continue Y OR N?
Y
```

```
Select DML Operation For Product Table...
1. Insert
2. Update
3. Delete
4. Select
5. Exit

Enter a choice:
3

Enter Product id:
2

Deleted Successfully!!!

Continue Y OR N?
y

Select DML Operation For Product Table...
1. Insert
2. Update
3. Delete
4. Select
5. Exit

Enter a choice:
4
ID Name Price Qty
---- -------- ----- ----
1 Sharpner 10.0 90

Continue Y OR N?
n
```