

## Shane's Computer Solution Repository

A place to record computer problems and solutions I encounter

### €3 Cheapest VPS in France

MVPS - Your trusted high quality  
VPS provider

MVPS LTD

## How to set up SQLite with JDBC in Eclipse on Windows



I'm working on a little project written in Java, and I want to have a simple database in it. I decided the easiest way to do it was with [SQLite](#), but it's been quite a while since I used SQLite in Java. So, in this post, I'm recording steps on what I had to do to use SQLite with JDBC in Eclipse on Windows, and I also have some really simple sample code showing how to use it.

### What's SQLite?

According to the SQLite home page:

SQLite is a C-language library that implements a small, fast, self-contained, high-reliability, full-featured, SQL database engine. SQLite is the most used database engine in the world. SQLite is built into all mobile phones and most computers and comes bundled inside countless other applications that people use every day.

OK, good enough for me.

### Wait a minute, SQLite is a C language library — I want to use it in Java!

Luckily, somebody has created a JDBC driver wrapped around the C language library, so that we can use it in Java programs. The code for the JDBC driver is available on GitHub at <https://github.com/xerial/sqlite-jdbc>, and there's extensive documentation linked from that page.

### Downloading the JDBC driver

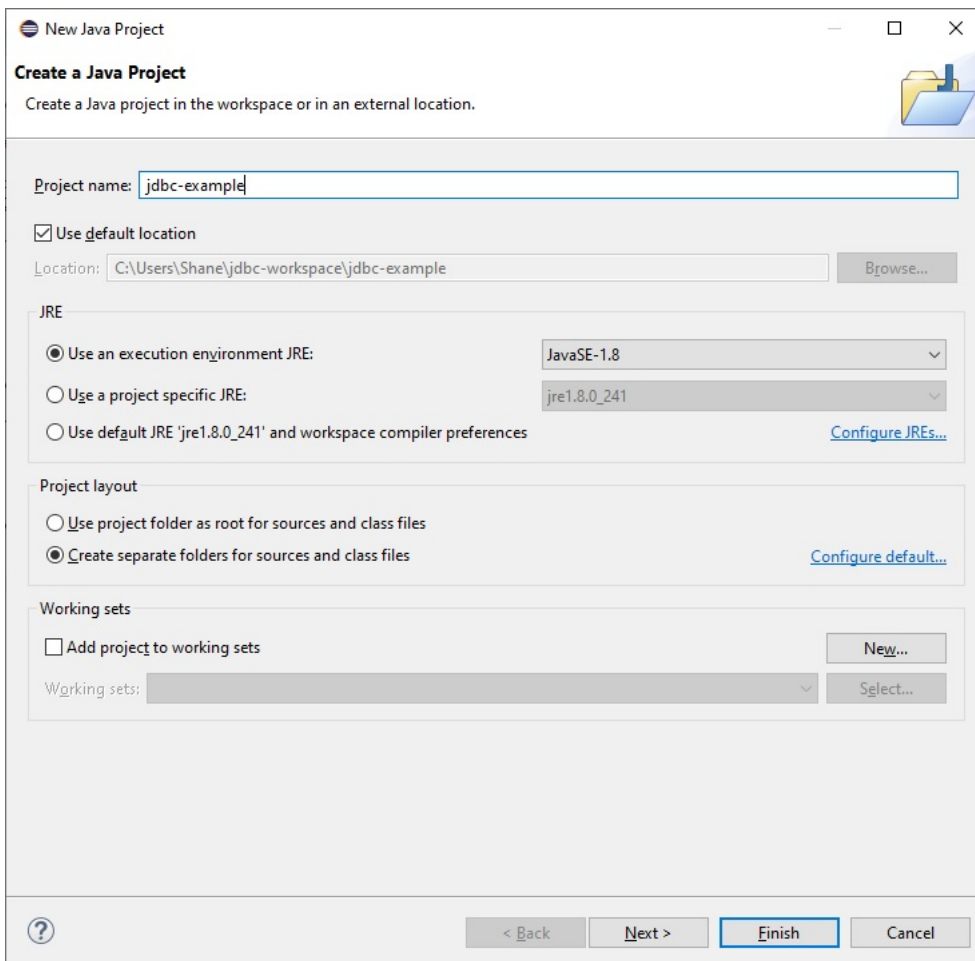
You can get the JDBC driver from a page on BitBucket: <https://bitbucket.org/xerial/sqlite-jdbc/downloads/>. At the time that I wrote this post, the latest version was 3.30.1 — download the file named sqlite-jdbc-3.30.1.jar, or a more recent version if there's one available when you read this.

We use cookies to ensure that we give you the best experience on our website. If you continue to use this site we will assume that you are happy with it.

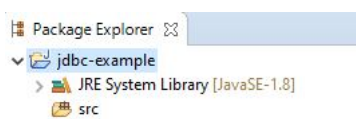
Ok [Privacy policy](#)

OK, I'm going to create a new Eclipse Java project to contain some examples of using the JDBC driver, and then I'll install the JDBC driver into that.

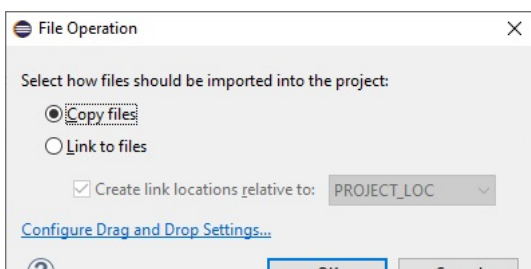
From the Eclipse menu, select File, then under that, select New, and then Java Project. This is the same way you create any Java project; using the JDBC driver makes no difference in this step. Give your project a name (I called mine jdbc-example), and select whatever options you want. I pretty much left mine at the default:



Click Finish (or Next if you want to do additional configuration), and you'll have a new empty Java project. Here's my project in the Eclipse Package Explorer:



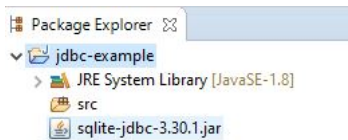
Now, you're going to want to put the JDBC driver that you downloaded somewhere that you can reference it. To make things easy, I'm just going to put it in my project folder. Copy the Jar file into the top level of your project — I just used the File Explorer to drag it from my Downloads folder and dropped it on the jdbc-example folder in the Eclipse Package Explorer. When I did that, Eclipse gave me the following window:



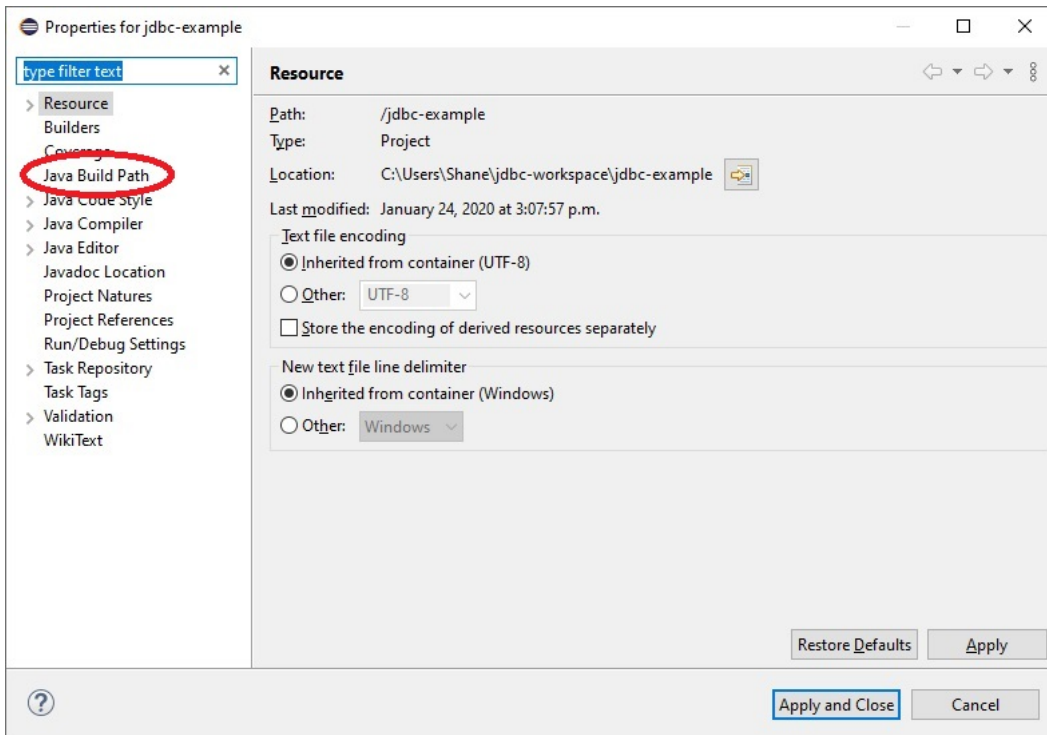
We use cookies to ensure that we give you the best experience on our website. If you continue to use this site we will assume that you are happy with it.

Ok Privacy policy

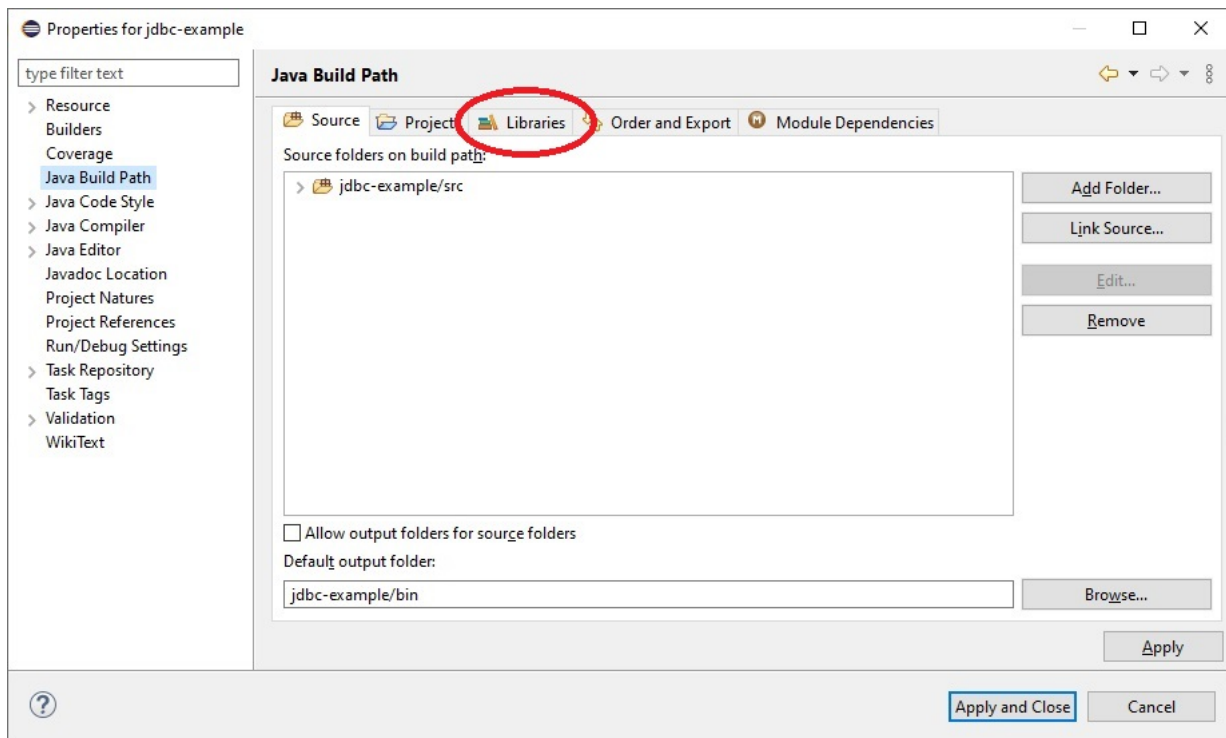
I want to make a copy of the Jar file, rather than linking to the file in my Downloads folder, so I left “Copy files” selected and pressed OK. When I did that, I can see that a copy of the Jar file is now in my project:



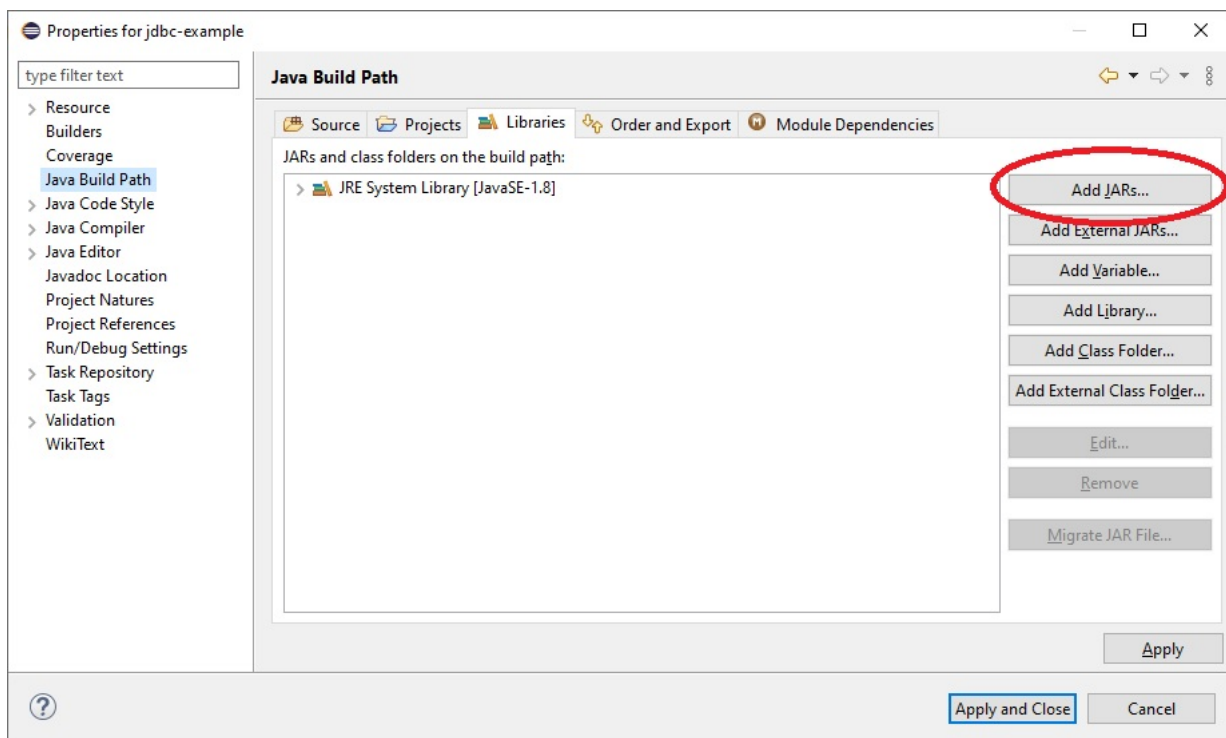
Note that I could put it anywhere; I can even reference it in my Downloads folder, but I like to keep my project files all together. Anyways, the next step is to tell Eclipse that I want to add that Jar file to my project's Java Build Path. Open the Project Properties by right-clicking on the jdbc-example project in the Package Explorer, then selecting Properties at the bottom of the pop-up menu (or alternatively, just click once on the jdbc-example project, then press Alt-Enter). You'll now get the project's Properties window, and it will look something like this (but without the red ellipse!):



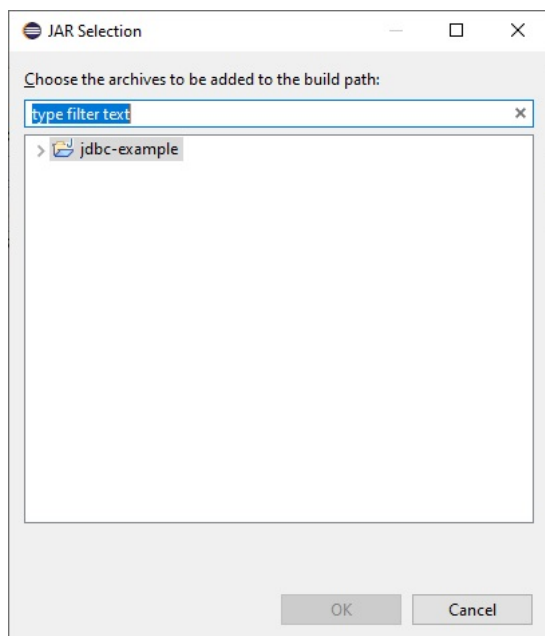
I want to change the Java Build Path. Click on Java Build Path on the left-hand side of the window (circled in red), and your window will now look like something like this (again, without the red ellipse):



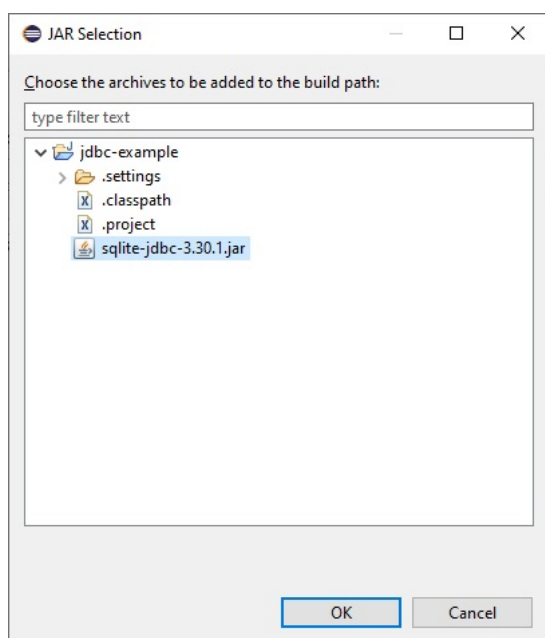
I want to change the Libraries, so on the right side of the window, select the Libraries pane (circled in red). Then, you'll get:



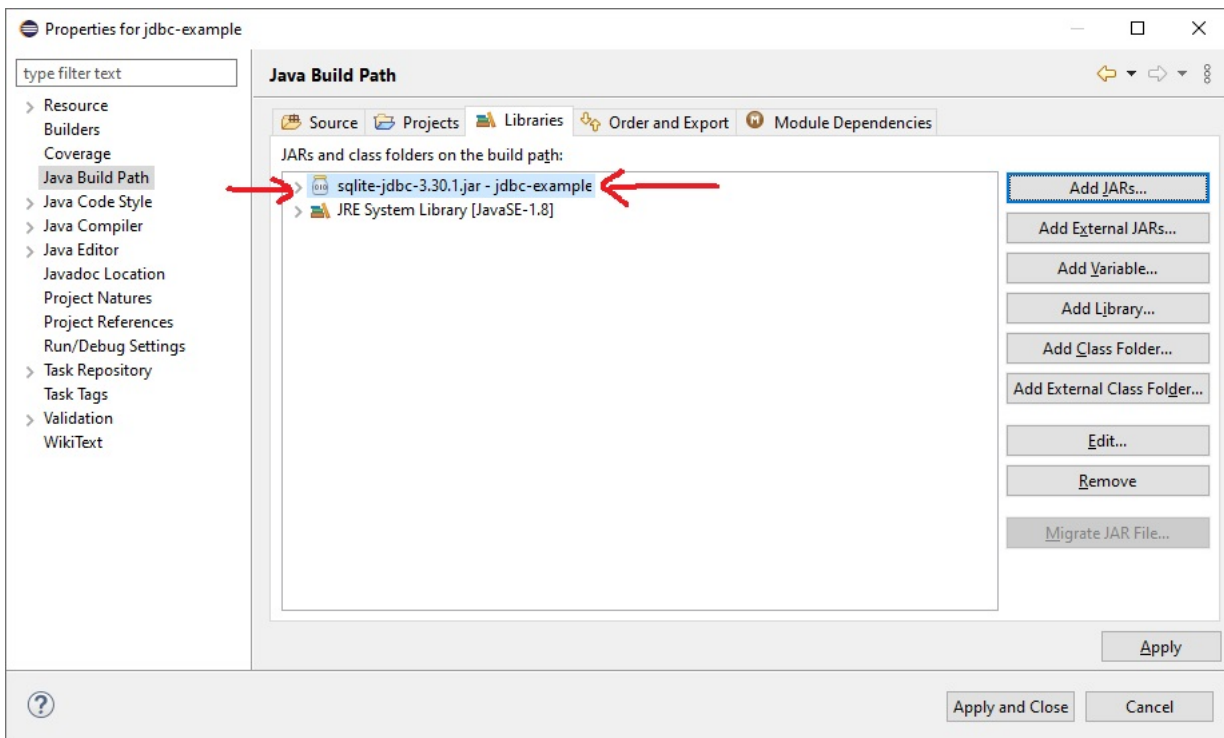
As you can probably guess, you'll now want to click on the "Add JARs..." button. You'll get the following pop-up window, listing the projects in your Eclipse workspace:



Expand your project:



Select sqlite-jdbc-3.30.1.jar, and press OK. You'll now see that file show up in the Libraries pane (funky hand-drawn arrows added by me!):



And you're now good to go! Click the "Apply and Close" button, and you're all set to use JDBC in your Java project in Eclipse!

## Is there any SQLite-JDBC API documentation?

Yup, there sure is. I found it at <https://www.javadoc.io/doc/org.xerial/sqlite-jdbc/3.30.1/index.html>.

## Some sample code

OK, you're probably thinking, that's great, but how do I use it? I've got some sample code and explanations below, but you can also check out the examples on the [JDBC driver's GitHub page](#) and the official "[JDBC Basics](#)" Java Tutorial from Oracle.

## Creating a database

The first thing you're going to want to do is create the database. You might already have one that you're trying to access, but if you don't, here's some code to do it for you.

Actually, all we're really going to do is to try to connect to a database. If it doesn't already exist, the SQLite JDBC driver will create one for us.



We use cookies to ensure that we give you the best experience on our website. If you continue to use this site we will assume that you are happy with it.

[Ok](#) [Privacy policy](#)

supplies a class called [SQLiteDataSource](#) that we'll use. Note that to use it, you'll need to import it from the `org.sqlite.SQLiteDataSource` package; you should be able to get Eclipse to do that for you when it gives you an error about not being able to resolve `SQLiteDataSource` to a type. Anyways, here's some code that creates the object:


```
SQLiteDataSource ds = new SQLiteDataSource();
```

We've now got a `DataSource` object, but it doesn't know where the database is! We can tell it that by using the [setUrl](#) method:

```
ds.setUrl("jdbc:sqlite:test.db");
```

So, this tells the driver we're going to connect to an SQLite database in a file called `test.db` in the current directory. If you want to specify a complete pathname, you'd do something like:

```
ds.setUrl("jdbc:sqlite:C:/users/shane/mydatabase.db");
```



## 5 - 1200 MW Solar Production

We help you to start your Photovoltaic Production Line  
- latest german technology!

Note that you'd use forward slashes, not backslashes as you might expect since we're doing this on Windows!

Once we've got the `DataSource` all set up, we need to tell it to create the database. Actually, what we're going to do is try to connect to the database; if it doesn't exist, the JDBC driver will create it for us. So, we need to create a [Connection](#) object. We can do that with:

```
Connection conn = ds.getConnection();
```

And that's it!

OK, I used a bit more code. Because setting up a `DataSource` might throw an exception, I want to wrap that in a try-catch block. And because a `Connection` can throw an [SQLException](#) when I try to get the connection, I'll do something similar. Note, though, that a `Connection` implements the [AutoCloseable](#) interface, so I can use the [try-with-resources statement](#) to make my code a little cleaner. So, here's some example code that will create an SQLite database in a file called `test.db`:

```
import java.sql.Connection;
import java.sql.SQLException;

import org.sqlite.SQLiteDataSource;
```

We use cookies to ensure that we give you the best experience on our website. If you continue to use this site we will assume that you are happy with it.

[Ok](#) [Privacy policy](#)

```

    SQLiteDataSource ds = null;

    try {
        ds = new SQLiteDataSource();
        ds.setUrl("jdbc:sqlite:test.db");
    } catch ( Exception e ) {
        e.printStackTrace();
        System.exit(0);
    }

    System.out.println( "Opened database successfully" );

    try ( Connection conn = ds.getConnection() ) {
    } catch ( SQLException e ) {
        e.printStackTrace();
        System.exit( 0 );
    }

    System.out.println( "Created database successfully" );
}
}

```

Global Trade Starts Here  
Alibaba.com



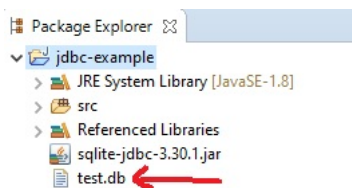
laptop

MZN 7,572

Visit Site

*A plea: this exception-catching code is horrendous. Although it's fine for an example, don't do it this way in real code. Think about what you're doing, and properly handle and clean up an exception!*

When you run this program, a file named test.db will be created in your project folder. If you refresh Eclipse's Package Explorer, you'll see it show up:



## Creating a database with a DriverManager

The old way of using JDBC was to use a [DriverManager](#) to create the connection. The code is simpler, but Oracle recommends using a DataSource for creating connections. But, if you want to do it the old way, here's sample code:

```

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

public class CreateDbWithDriverManagerExample {

    public static void main(String[] args) {
        try (Connection conn =
            DriverManager.getConnection("jdbc:sqlite:test.db");

```

We use cookies to ensure that we give you the best experience on our website. If you continue to use this site we will assume that you are happy with it.

Ok [Privacy policy](#)



```
    }  
  
    System.out.println("Opened database successfully");  
}  
  
}
```

## Creating a table

Creating a table is a nice example, because you will use the same concepts to perform many other database operations, other than reading from the database. Anyways, you start by creating a `DataSource` object and a `Connection` object. Once you've got a `Connection` object, you can create a [Statement](#) object, and you'll use that `Statement` object to execute the command. To create the `Statement` object from a `Connection` object named `conn`, do the following:

```
Statement stmt = conn.createStatement();
```

Again, a `Statement` implements `AutoCloseable`, so you can use a try-with-resources statement to simplify your code (see the full example code below).

Once you've got a `Statement`, there are three methods that you can call to execute database commands:

- [executeUpdate\(\)](#), which is used to update the database. It returns the number of rows in the database that were affected by the `Statement`. This is what you'll use when you execute a `CREATE`, `INSERT`, `DELETE`, or `UPDATE` statement.
- [executeQuery\(\)](#), which is used when you are making a query against the database. It returns a [ResultSet](#) object, which contains a number of results. This is what you'll use when you execute a `SELECT` statement.
- [execute\(\)](#), which is used when you are expecting one or more `ResultSet` objects to be returned. I won't be using this in this post, because I'll only return a single `ResultSet` object — `ResultSet` objects can contain more than one result.

OK, so if we're going to create a table, we'll want to use `executeUpdate`. Here's the SQL code to create a table I'll use in these examples:

```
CREATE TABLE IF NOT EXISTS test  
( ID INTEGER PRIMARY KEY,  
  NAME TEXT NOT NULL );
```

To run the command, pass it into `executeUpdate()` as a `String`. For my example code, I'm going to create a `String` object containing the command, then pass that `String` in. I could just pass a `String` constant in, but I thought my code was a little more readable this way.

```
Statement stmt = conn.createStatement();
int rv = stmt.executeUpdate( query );
```

Note: rv is short for return value. Here's my full code example:

```
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;

import org.sqlite.SQLiteDataSource;

public class CreateTableExample {

    public static void main(String[] args) {
        SQLiteDataSource ds = null;

        try {
            ds = new SQLiteDataSource();
            ds.setUrl("jdbc:sqlite:test.db");
        } catch ( Exception e ) {
            e.printStackTrace();
            System.exit(0);
        }
        System.out.println( "Opened database successfully" );

        String query = "CREATE TABLE IF NOT EXISTS test ( " +
            "ID INTEGER PRIMARY KEY, " +
            "NAME TEXT NOT NULL )";

        try ( Connection conn = ds.getConnection();
            Statement stmt = conn.createStatement(); ) {
            int rv = stmt.executeUpdate( query );
            System.out.println( "executeUpdate() returned " + rv );
        } catch ( SQLException e ) {
            e.printStackTrace();
            System.exit( 0 );
        }
        System.out.println( "Created database successfully" );
    }
}
```

When you run it, it creates the table. Note that rv will have the value 0, because no rows were updated.

## Inserting rows into the table

Inserting rows into the table is basically the exact same code, you just change the query String:

We use cookies to ensure that we give you the best experience on our website. If you continue to use this site we will assume that you are happy with it.

[Ok](#) [Privacy policy](#)

When you call `executeUpdate()`, it will return the value 1, because you have updated 1 row.

Here's a snippet of my example code. I'm actually inserting two rows, one at a time. In my example, you can see they are both wrapped inside the same try-with-resources statement:

```
System.out.println( "Attempting to insert two rows into test table" );

String query1 = "INSERT INTO test ( NAME ) VALUES ( 'Shane' )";
String query2 = "INSERT INTO test ( NAME ) VALUES ( 'Sharon' )";

try ( Connection conn = ds.getConnection();
      Statement stmt = conn.createStatement(); ) {
    int rv = stmt.executeUpdate( query1 );
    System.out.println( "1st executeUpdate() returned " + rv );

    rv = stmt.executeUpdate( query2 );
    System.out.println( "2nd executeUpdate() returned " + rv );
} catch ( SQLException e ) {
    e.printStackTrace();
    System.exit( 0 );
}
```

## Updating and deleting rows

Once again, when updating and deleting rows, you use the same code, just change the query string. Examples:

```
String query = "UPDATE test SET NAME = 'Poopsie'" +
               " WHERE NAME = 'Sharon'";
```

and

```
String query = "DELETE FROM test WHERE NAME = 'Shane'";
```

## Deleting a table

Deleting a table works the same way, just a different query String:

```
String query = "DROP TABLE IF EXISTS test";
```

## Querying a table

To query a table, create a query String, but pass it into `executeQuery()` rather than `executeUpdate()`:

```
String query = "SELECT * FROM test";

ResultSet rs = stmt.executeQuery(query);
```

We use cookies to ensure that we give you the best experience on our website. If you continue to use this site we will assume that you are happy with it.

[Ok](#) [Privacy policy](#)

A `ResultSet` object is returned. This contains a number of results. See the Java Tutorial “[Retrieving and Modifying Values from Result Sets](#)” for details, but here’s a simple case where I loop through all the results that come back in the `ResultSet` and display them:

```
while ( rs.next() ) {  
    int id = rs.getInt( "ID" );  
    String name = rs.getString( "NAME" );  
  
    System.out.println( "Result: ID = " + id + ", NAME = " + name );  
}
```

The [next\(\)](#) method returns true if there are more results in the `ResultSet` to be processed. To get the actual values from the result, you use the various `get*()` methods, depending on the type of the row. In my example table, the ID field is an `INTEGER`, and the NAME field is `TEXT`, so I use [getInt\(\)](#) for the ID and [getString\(\)](#) for the NAME.

Here’s my full example code:

```
import java.sql.Connection;  
import java.sql.ResultSet;  
import java.sql.SQLException;  
import java.sql.Statement;  
  
import org.sqlite.SQLiteDataSource;  
  
public class SelectRowsExample {  
  
    public static void main(String[] args) {  
        SQLiteDataSource ds = null;  
  
        try {  
            ds = new SQLiteDataSource();  
            ds.setUrl("jdbc:sqlite:test.db");  
        } catch ( Exception e ) {  
            e.printStackTrace();  
            System.exit(0);  
        }  
  
        System.out.println( "Opened database successfully" );  
  
        System.out.println( "Selecting all rows from test table" );  
        String query = "SELECT * FROM test";  
  
        try ( Connection conn = ds.getConnection();  
              Statement stmt = conn.createStatement(); ) {  
            ResultSet rs = stmt.executeQuery(query);  
  
            while ( rs.next() ) {  
                int id = rs.getInt( "ID" );  
                String name = rs.getString( "NAME" );  
            }  
        }  
    }  
}
```

We use cookies to ensure that we give you the best experience on our website. If you continue to use this site we will assume that you are happy with it.

[Ok](#) [Privacy policy](#)

```
        } catch ( SQLException e ) {  
            e.printStackTrace();  
            System.exit( 0 );  
        }  
    }  
}
```

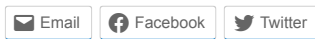
## Final words

So, there's the basics of how you set up the SQLite JDBC driver, and some simple examples of using it. Refer to the Oracle documentation for more detailed examples, but these examples should get you through the basics. Have fun!

## Where can I get a copy of the sample code?

I'm glad you asked. I put a copy up on BitBucket. You can find it at <https://bitbucket.org/shaneroo/jdbc-example>.

Share this:



Shane / January 24, 2020 / eclipse, tutorial / eclipse, Java, jdbc, sqlite, windows

---

## 2 thoughts on “How to set up SQLite with JDBC in Eclipse on Windows”

---

**Rick**

September 16, 2020 at 2:08 PM

I like your sample – do you have anything where you output the data to a java form

---

**Shane** 

September 21, 2020 at 2:16 PM

No, but you should be able to use the sample “Querying a table” code, then take what’s in the ResultSet and stick that into your form.

This site uses Akismet to reduce spam. [Learn how your comment data is processed.](#)

Shane's Computer Solution Repository / [Privacy Policy](#) / Proudly powered by WordPress

We use cookies to ensure that we give you the best experience on our website. If you continue to use this site we will assume that you are happy with it.

[Ok](#) [Privacy policy](#)