Connecting to a MySQL Database using Connector/J JDBC Driver

by Faisal Khan.

Overview
In this tutorial following topics will be covered:

- What are Database URLs in JDBC?
- Why and how to specify a JDBC Driver name?
- How to create a connection to a Database?
- An example on how to connect to a MySQL Database?

Note: This tutorial assumes you've correctly setup MySQL database and Connector/J JDBC driver. If not then consult "Installing and Configuring MySQL Database and Connector/J JDBC Driver on Microsoft Windows".

What are Database URLs in JDBC?
URL stands for "Uniform Resource Locator". You will be familiar with HTTP URLs that you normally use to access a web site e.g. http://www.stardeveloper.com. URLs are used to identify a resource using a unique name.

Same goes for database URLs in JDBC. JDBC requires that all database connection strings should be represented by URLs. The URLs used in JDBC have following structure:

```
jdbc:subprotocol:subname
```

In HTTP you begin a URL with the protocol name i.e. http:, similarly in JDBC driver URLs, you start the URL with protocol name i.e. jdbc:. Next `subprotocol` represents the database you want to connect to e.g. mysql, oracle, odbc etc. While `subname` provides additional information on how and where to connect.

Tip: If you are familiar with ASP/ASP.NET, a database URL in JDBC is quite similar to a connection string used in an ASP environment to connect to a database.

Examples of Database URLs
Following are some examples of JDBC database URLs:

- `jdbc:odbc:dsn_name;UID=your_uid;PWD=your_pwd` - JDBC-ODBC Bridge Driver URL.
- `jdbc:oracle:thin:@machine_name:port_number:instance_name` - Orace Type 4 JDBC Driver.
Why and how to specify a JDBC Driver name?
Next thing you need to know besides the database URL is the full class name of your JDBC driver e.g. `com.mysql.jdbc.Driver` in case of MySQL Connector/J JDBC driver. The name of the driver is a requirement and is not optional.

**Note:** In ASP/ASP.NET both database URL and driver name are part of a single connection string, while Java separates them for greater flexibility.

You can tell JVM about what driver/s to use by using one of the following methods:

- To load the driver/s at JVM startup, specify the driver/s in `jdbc.drivers` system property like this:

  ```
  java -Djdbc.drivers=com.mysql.jdbc.Driver YourJavaProgram
  ```

- To explicitly load the driver, use `Class.forName()` method in your code like this:

  ```
  Class.forName("com.mysql.jdbc.Driver").newInstance();
  ```

**Note:** In above examples, "com.mysql.jdbc.Driver" is the name of the JDBC driver that you want to load.

The example discussed in this tutorial makes use of the second option discussed above.

How to create a connection to a Database?
To create a connection to a database, you will have to use `java.sql.DriverManager`'s `getConnection()` method. This method takes as an argument the database URL (that we discussed earlier) you want to connect to. It then internally finds the appropriate driver which has been loaded in the JVM and then delegates the work of creating the connection to that driver.

An example on how to connect to a MySQL Database?
After learning the theory behind connecting to a database, we'll now move on to create a Java program which will connect to a MySQL database running on your local system.

**Note:** This tutorial assumes you've correctly setup MySQL database and Connector/J JDBC driver. If not then consult "Installing and Configuring MySQL Database and Connector/J JDBC Driver on Microsoft Windows".

```java
package com.stardeveloper.example;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

JdbcExample2.java
Create a new Java source file and save it as `JdbcExample2.java`. Copy/paste following code in it:
```
public class JdbcExample2 {

    public static void main(String args[]) {
        Connection con = null;

        try {
            Class.forName("com.mysql.jdbc.Driver").newInstance();
            con = DriverManager.getConnection("jdbc:mysql:///test",
            "root", "secret");

            if(!con.isClosed())
                System.out.println("Successfully connected to " +
                "MySQL server using TCP/IP...");
        } catch(Exception e) {
            System.err.println("Exception: " + e.getMessage());
        } finally {
            try {
                if(con != null)
                    con.close();
            } catch(SQLException e) {
            }
        }
    }

    Explanation
    Above Java program tries to connect to "test" database installed on your local MySQL server.
    "test" database is installed on all MySQL servers by default so that is why I am using this
database to connect to, otherwise as far as this tutorial is concerned, it won't be accessing any	tables in that database.

    On the next page, we will look into this code and learn what it does, step-by-step. We will then
execute this code and display the status message on the console.

    Let us see the code now. The first statement describes that this Java program belongs to	com.stardeveloper.example package:

    package com.stardeveloper.example;

    Next, we import 3 classes from java.sql package. Connection represents a connection to a
database, DriverManager manages JDBC drivers and is used to create connections to databases and
SQLException is an exception class which gets thrown in case any error ocurrs in the program.

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

    Next we give a name to our Java program; JdbcExample2.

    public class JdbcExample2 {
Next we stuff all the code in the main() method of our Java program.

**Note:** main() method provides an entry point in your Java program. Your program starts executing from here.

```java
public static void main(String args[]) {
    ...
}
```

We declare a local variable to hold our java.sql.Connection object that we'll retrieve from java.sql.DriverManager later on.

```java
Connection con = null;
```

We encapsulate our JDBC code in a try/catch/finally block. This is to ensure that if any runtime error gets thrown we catch it and display it gracefully to the user.

```java
try {
    ...
} catch (Exception e) {
    System.err.println("Exception: " + e.getMessage());
} finally {
    ...
}
```

Now we take the first step in establishing a connection to our MySQL database by loading the driver explicitly using Class.forName() method like this:

```java
Class.forName("com.mysql.jdbc.Driver").newInstance();
```

Once the JDBC driver has been loaded in the JVM, we retrieve a connection to our MySQL database running on the local system using DriverManager.getConnection() method. The first argument to this method is complete database URL, next parameter is the user name and last parameter is the password.

**Note:** You'd want to change the password described here with the actual password that you've setup for your MySQL 'root' account.

```java
con = DriverManager.getConnection("jdbc:mysql:///test", "root", "secret");
```

If everything goes on well, above 2 statements are enough to successfully obtain a connection to our MySQL database. We then check to make sure we are properly connected and display a success message if we are.

```java
if(!con.isClosed())
    System.out.println("Successfully connected to " +
                   "MySQL server using TCP/IP...");
```
Now to gracefully close the connection, we execute the `Connection.close()` method in the finally block of our try/catch/finally statement.

```java
try {
    if (con != null)
        con.close();
} catch (SQLException e) {}
```

**Note:** The `Connection.close()` method can also throw `SQLException` so that is why we encapsulate this `close()` method in a separate try/catch block.

Hoa! this is all that is required to successfully connect to a running instance of MySQL server on your local system and print a success message. Let us move on and compile/run our Java program.

**Compiling JdbcExample2.java**
To compile the JdbcExample2.java program to it's class file and place it according to it's package statements, open command prompt and cd (change directory) into the folder containing JdbcExample2.java, then execute this command:

```
javac -d . JdbcExample2.java
```

**Note:** The zip file that comes with this tutorial contains all the source and class files required to successfully run this Java program.

If all goes well you should get a new Java class under the current folder with a directory structure like `com/stardeveloper/example/JdbcExample2.class`.

**Running JdbcExample2.java**
Before we try to execute our Java program, make sure that MySQL server is running on your local system. If it not running then start the MySQL database server and move on.

To run this program execute following command from the command prompt from the same folder where JdbcExample2.java is residing:

```
java com.stardeveloper.example.JdbcExample2
```
If all goes well you should see a success message printed on your console. Here is how it looked when I executed this program on my system:

![Command Prompt - Running JdbcExample2 Java Program](image)

Congratulations! you've successfully coded, compiled and run your first JDBC program which connects to a MySQL server running on your local system using Connector/J JDBC driver.

**Summary**

In this tutorial we learned the basics of connecting to a database (MySQL) using a JDBC driver (Connector/J). We learned what are database URLs and why/how to specify JDBC driver names. We then moved on to write a simple Java program which connects to a MySQL database and prints a success message.

In next article we will learn on how to create and execute SQL statements against a database using JDBC.