MySQL Connectivity

MySQL connectivity is established in Java using classes and methods supplied in the Java Database Connectivity (JDBC) API. This defines a standard set of interfaces by which a Java application may connect to a wide range of SQL databases, as well as other data sources such as spreadsheets. These interfaces also define methods for performing common tasks such as updating & querying the database.

Step 1: Obtaining the MySQL JDBC Driver

A database driver is a software module designed to interact with a particular Database Management System (DBMS) which may or not be written in Java. A driver for a particular DBMS is typically written by the product vendor. If the driver is not written in Java itself, a set of wrapper classes will be provided which will interact with the non-Java code on our behalf via the Java Native Interface (JNI) – these classes will conform to the JDBC interfaces in order that these classes may be utilized in a uniform manner. Obviously, if the driver itself is written in Java, the process is more straightforward, though as developers utilising existing drivers we need not concern ourselves with how these drivers are implemented which is the beauty of the JDBC API. There are a number of means by which a Java application may connect to a MySQL database; here we present the simplest of these in a series of steps.

Assuming that both Java SE and MySQL are installed on your own machine or accessible in some manner, the first step in establishing MySQL connectivity involves obtaining the appropriate JDBC driver for your DBMS. Note that because the vendors of MySQL provide support for the use of MySQL from a plethora of programming languages and many of these have different APIs for establishing connectivity, the vendors of MySQL have dubbed these as 'connectors'. Inside the **Connector/J** folder you should find a .ZIP file called **mysql-connector-java-5.1.26-bin.jar.** This file contains a large number of Java classes essential for database connectivity. These classes need to be made available to your Java programs. To do this, follow the steps below:

In order to utilise this file, you must ensure that it is imported into the *classpath* of your Java application.

Figure 1: Adding the MySQL JDBC Driver to a Netbeans project classpath

Step 2: Loading the JDBC Driver class

Every driver implementation must provide a class which implements the **`java.sql.Driver`** interface and this class, as has been indicated, is usually provided by the product vendor. Java, however, does come with a class named **`DriverManager`** which is tasked with utilising the appropriate driver in order to connect to the specified DBMS. In

fact, the **`DriverManager`** instance will try to load as many drivers as it can find and then for any given connection request, it will ask each driver in turn to try to connect to the target URL.

We first of all need to load the implementation of the **`Driver`** interface supplied by the downloaded driver (**JAR** file). Note that when a **`Driver`** class is loaded, it should create an instance of itself and register it with the **`DriverManager`** which in turn will utilise this to connect to the DBMS when requested.

A programmer can load the driver as follows:

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

In the above, **`com.mysql.jdbc.Driver`** is the name of the class implementing the **`java.sql.Driver`** interface.

Note that JDBC 4.0 contains a feature known as driver *auto-loading* which means that we are able to omit the above line, however we include this information for the purposes of backward compatibility. For more information on this topic see Chapter 4 in Horstmann's*Core Java Volume 2 – Advanced Features*.

Step 3: Getting a Connection

A connection to the database is represented in Java using a **`java.sql.Connection`** instance which may be obtained by invoking the **`getConnection()`** method of the **`DriverManager`** class. In fact, multiple implementations of this method are provided by the DriverManager class as the method is overloaded allowing connection information to be supplied in a variety of methods. In this guide, we will look at the version which accepts three String objects as parameters, these being:

- The JDBC URL of the database to connect to;
- The username to authenticate to the database;
- The password to authenticate to the database.

A JDBC URL is much like any other URL you will have encountered previously, being used in order to connect to a particular resource, in this case, a database. Unfortunately, JDBC URLs are non-standardised which means that they are of a different format for every type of DBMS we wish to use, the only thing they have in common being that they must begin with the prefix 'jdbc:'.

Note that MySQL JDBC URLs can become quite complex and for the gory details consult the MySQL Reference Manual. Luckily, however, for purposes of this lab session the form below will be sufficient:

- `jdbc:mysql://[host][:port]/[database]`

Note also that the default port for connecting to an instance of a MySQL DBMS is 3306 so the port number is optional but probaly best to put in as in the example below:

- *jdbc*:*mysql*://localhost:3306/test

In the above, we are assuming that the DBMS is running on the local machine, the port number being used is the default and that the name of the database is 'test'.

Soputting all this together we can write some Java to connect to a MySQL DBMS.

```
String url = "jdbc:mysql://localhost:3306/test";

String username = "myUsername";

String password = "myPassword";

// Load driver class for MySQL.

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

// Get a connection to the database called test.

Connection connection

      = DriverManager.getConnection(url, username, password);
```

You should note that the method getConnection() may throw a "java.sql.SQLException" in the event of error, therefore you will also require a try… catch block.

As you will realise these notes are written on the assumption that you are using a local version of MySQL installed on a computer in the Dean St site or have installed it on your own machine at home or laptop.

Note: Although students have access to an SQL server in the labs it is recommended that you go through the installation process yourself as this provides some valuable experience setting up libraries and services within an IDE environment.

Step 4: Interacting with the Database

Having established a connection to the MySQL server it is then possible to interact with the database specified in the URL. The fragment below illustrates this idea:

```
Statement stat = connection.createStatement();
try{
      stat.execute("CREATE TABLE Test(idno INT);");
      stat.execute("INSERT INTO Test VALUES (123)");

      ResultSet result = stat.executeQuery("SELECT * FROM
Test");
      result.next();
      System.out.println(result.getString("idno"));

      stat.execute("DROP TABLE Test");
```

```
}
finally
{
      connection.close();
}
```

Note that you should always close the connection in a **`finally`** block when you are done using it in order to avoid tying up system resources.