JDBC DATABASE CONNECTIONS

After you've installed the appropriate driver, it's time to establish a database connection using JDBC.

The programming involved to establish a JDBC connection is fairly simple. Here are these simple four steps:

- **Import JDBC Packages:** Add **import** statements to your Java program to import required classes in your Java code.
- **Register JDBC Driver:** This step causes the JVM to load the desired driver implementation into memory so it can fulfill your JDBC requests.
- **Database URL Formulation:** This is to create a properly formatted address that points to the database to which you wish to connect.
- **Create Connection Object:** Finally, code a call to the *DriverManager* object's *getConnection( )* method to establish actual database connection.

Import JDBC Packages:

The **Import** statements tell the Java compiler where to find the classes you reference in your code and are placed at the very beginning of your source code.

To use the standard JDBC package, which allows you to select, insert, update, and delete data in SQL tables, add the following *imports* to your source code:

```
import java.sql.* ;  // for standard JDBC programs
import java.math.* ; // for BigDecimal and BigInteger support
```

Register JDBC Driver:

You must register the your driver in your program before you use it. Registering the driver is the process by which the Oracle driver's class file is loaded into memory so it can be utilized as an implementation of the JDBC interfaces.

You need to do this registration only once in your program. You can register a driver in one of two ways.

Approach (I) - Class.forName():

The most common approach to register a driver is to use Java's **Class.forName()** method to dynamically load the driver's class file into memory, which automatically registers it. This method is preferable because it allows you to make the driver registration configurable and portable.

The following example uses Class.forName( ) to register the Oracle driver:

```
try {
   Class.forName("oracle.jdbc.driver.OracleDriver");
}
catch(ClassNotFoundException ex) {
   System.out.println("Error: unable to load driver class!");
   System.exit(1);
}
```

You can use **getInstance()** method to work around noncompliant JVMs, but then you'll have to code for two extra Exceptions as follows:

```
try {
   Class.forName("oracle.jdbc.driver.OracleDriver").newInstance();
}
catch(ClassNotFoundException ex) {
   System.out.println("Error: unable to load driver class!");
   System.exit(1);
```

```
catch(IllegalAccessException ex) {
   System.out.println("Error: access problem while loading!");
   System.exit(2);
catch(InstantiationException ex) {
   System.out.println("Error: unable to instantiate driver!");
   System.exit(3);
}
```

Approach (II) - DriverManager.registerDriver():

The second approach you can use to register a driver is to use the static **DriverManager.registerDriver()** method.

You should use the *registerDriver()* method if you are using a non-JDK compliant JVM, such as the one provided by Microsoft.

The following example uses registerDriver() to register the Oracle driver:

```
try {
   Driver myDriver = new oracle.jdbc.driver.OracleDriver();
   DriverManager.registerDriver( myDriver );
}
catch(ClassNotFoundException ex) {
   System.out.println("Error: unable to load driver class!");
   System.exit(1);
}
```

Database URL Formulation:

After you've loaded the driver, you can establish a connection using the **DriverManager.getConnection()** method. For easy reference, let me list the three overloaded DriverManager.getConnection() methods:

- getConnection(String url)
- getConnection(String url, Properties prop)
- getConnection(String url, String user, String password)

Here each form requires a database **URL**. A database URL is an address that points to your database.

Formulating a database URL is where most of the problems associated with establishing a connection occur.

Following table lists down popular JDBC driver names and database URL.

RDBMS	JDBC driver name	URL format
MySQL	com.mysql.jdbc.Driver	**jdbc:mysql:**//hostname/ databaseName
ORACLE	oracle.jdbc.driver.OracleDriver	**jdbc:oracle:thin:**@hostname:port Number:databaseName
DB2	COM.ibm.db2.jdbc.net.DB2Driver	**jdbc:db2:**hostname:port Number/databaseName
Sybase	com.sybase.jdbc.SybDriver	**jdbc:sybase:Tds:**hostname: port Number/databaseName

All the highlighted part in URL format is static and you need to change only remaining part as per your database setup.

Create Connection Object:

Using a database URL with a username and password:

I listed down three forms of **DriverManager.getConnection()** method to create a connection object. The most commonly used form of getConnection() requires you to pass a database URL, a *username*, and a *password*:

Assuming you are using Oracle's **thin** driver, you'll specify a host:port:databaseName value for the database portion of the URL.

If you have a host at TCP/IP address 192.0.0.1 with a host name of amrood, and your Oracle listener is configured to listen on port 1521, and your database name is EMP, then complete database URL would then be:

```
jdbc:oracle:thin:@amrood:1521:EMP
```

Now you have to call getConnection() method with appropriate username and password to get a **Connection** object as follows:

```
String URL = "jdbc:oracle:thin:@amrood:1521:EMP";
String USER = "username";
String PASS = "password"
Connection conn = DriverManager.getConnection(URL, USER, PASS);
```

Using only a database URL:

A second form of the DriverManager.getConnection( ) method requires only a database URL:

```
DriverManager.getConnection(String url);
```

However, in this case, the database URL includes the username and password and has the following general form:

```
jdbc:oracle:driver:username/password@database
```

So the above connection can be created as follows:

```
String URL = "jdbc:oracle:thin:username/password@amrood:1521:EMP";
Connection conn = DriverManager.getConnection(URL);
```

Using a database URL and a Properties object:

A third form of the DriverManager.getConnection( ) method requires a database URL and a Properties object:

```
DriverManager.getConnection(String url, Properties info);
```

A Properties object holds a set of keyword-value pairs. It's used to pass driver properties to the driver during a call to the getConnection() method.

To make the same connection made by the previous examples, use the following code:

```
import java.util.*;

String URL = "jdbc:oracle:thin:@amrood:1521:EMP";
Properties info = new Properties( );
info.put( "user", "username" );
info.put( "password", "password" );

Connection conn = DriverManager.getConnection(URL, info);
```

Closing JDBC connections:

At the end of your JDBC program, it is required explicitly close all the connections to the database to end each database session. However, if you forget, Java's garbage collector will close the connection when it cleans up stale objects.

Relying on garbage collection, especially in database programming, is very poor programming practice. You

should make a habit of always closing the connection with the close() method associated with connection object.

To ensure that a connection is closed, you could provide a finally block in your code. A *finally* block always executes, regardless if an exception occurs or not.

To close above opened connection you should call close() method as follows:

```
conn.close();
```

Explicitly closing a connection conserves DBMS resources, which will make your database administrator happy.

For a better understanding, I would suggest to study our JDBC - Sample Code.