CICLO 2

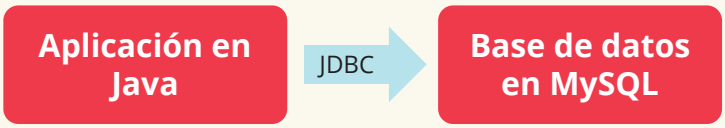
[FORMACIÓN POR CICLOS]

Introducción a JDBC

Ingeni@ Soluciones TIC

UNIVERSIDAD DE ANTIOQUIA
Facultad de Ingeniería

Como sabemos, la persistencia de información en aplicaciones de todo tipo y en todos los lenguajes se puede hacer mediante diferentes mecanismos. Los más populares son: el uso de archivos en un disco de almacenamiento y las bases de datos. Aquí, se hace necesaria la interacción entre un lenguaje de programación y una base de datos. Lo cual, normalmente, se logra mediante conexiones a un sistema de gestión de bases de datos (SGBD) específico. Lo anterior se logra mediante una arquitectura o estructura cliente-servidor. En esta, una aplicación se debe conectar a un servidor de bases de datos y puede realizar consultas a este para guardar, gestionar u obtener información de este. Para el caso de las bases de datos relacionales en general y las bases de datos MySQL en particular, el puente entre ambos mundos (cliente y servidor) es JDBC.



JDBC (siglas de *Java Database Connectivity*) es una API estándar de Java para gestionar la conexión entre el lenguaje Java y una base de datos, independientemente del SGBD elegido. Se usa muy frecuentemente para conectarse a una gran cantidad de SGBD de bases de datos relacionales, y para todos ellos funciona de una manera muy similar. JDBC permite:

- Conectarse a una fuente de datos, específicamente una base de datos.
- Enviar consultas a la base de datos.
- Recuperar y procesar los datos resultantes de una consulta.

Clases Clave en JDBC

JDBC es una API robusta, por lo que cuenta con un número importante de clases. Sin embargo, las clases e interfaces clave para las funciones más sencillas en JDBC son las siguientes:

• `java.sql.Connection`: Representa la conexión de la aplicación de Java con la base de datos. Así que, para realizar consultas a una base de datos, será necesario crear un objeto de este tipo y comunicarse con la base de datos a través de él.
• `java.sql.SQLException`: Clase correspondiente a las excepciones que se pueden presentar al comunicarse con una base de datos mediante JDBC.
• `java.sql.Statement`: Los objetos de este tipo representan consultas SQL que realizamos a la base de datos.
• `java.sql.ResultSet`: Objeto que contiene los registros recuperados de la base de datos mediante una consulta con un objeto `Statement` o `PreparedStatement`. Por lo general, podremos iterar sobre el `ResultSet` para operar sobre cada uno de los registros recuperados.
• `java.sql.PreparedStatement`: Permite usar sentencias preparadas, útiles, por ejemplo, para consultas con datos ingresados por el usuario.

Importación del Driver de MySQL

JDBC provee un conjunto de clases e interfaces que definen cómo nos conectamos y cómo consultamos en una base de datos, independientemente del SGBD que se trate. Sin embargo, es necesario importar dentro del proyecto de Java en desarrollo el *driver* respectivo, que sí depende del SGBD que vayamos a utilizar. Esto quiere decir que los tipos y métodos definidos en JDBC se pueden usar para cualquier SGBD (MySQL, SQL Server y Oracle, por ejemplo), pero será necesario importar el *driver* del SGBD de nuestra elección para que dichos métodos funcionen. Existen dos formas de importar dicho *driver* para el caso de MySQL.

• Una forma es descargar e importar el *driver* de MySQL dentro del proyecto en el IDE. Esta es la primera elección cuando no se trata de un proyecto con Maven o *Gradle*. Lo más conveniente es descargar el archivo .JAR correspondiente al *driver* e importarlo al proyecto dentro del IDE en la sección de librerías o de dependencias. En el caso de MySQL, el *driver* lo podemos descargar aquí: https://dev.mysql.com/downloads/connector/j/
• La otra forma, si estamos trabajando en un proyecto con *Maven*, es agregar la dependencia de MySQL en el archivo pom.xml del proyecto. La dependencia se puede encontrar aquí: https://mvnrepository.com/artifact/mysql/mysql-connector-java/8.0.26

Al momento de importar el *driver* de MySQL a un proyecto de Java, debemos verificar que la versión del driver sea compatible con la versión de MySQL con la que estemos trabajando y, en algunos casos, con nuestra versión del JDK (si esta es muy antigua).

Uso de JDBC

El uso de JDBC consta, por lo general, de los siguientes pasos:

1. Importar los paquetes necesarios. Normalmente hablamos de los paquetes de JDBC y de paquetes específicos del *driver* del SGBD de la base de datos a la que vamos a acceder.
2. Abrir una conexión a dicha base de datos, mediante un objeto del tipo `Connection`.
3. Ejecutar la consulta deseada en la base de datos, mediante un objeto del tipo `Statement` o `PreparedStatement`.
4. Extraer y usar los datos recuperados mediante la consulta, iterando sobre un objeto `ResultSet`. Obviamente, lo anterior aplica en consultas del tipo `SELECT`, no siendo así en aquellas de inserción, actualización o eliminación de registros.
5. Cerrar todas las conexiones abiertas (de ser necesario).

Ejemplo de Uso

A continuación, mostramos un ejemplo de cómo consultar una base de datos MySQL desde Java usando JDBC con el *driver* respectivo[1]. Para ejecutarlo, es necesario tener apropiadamente instalado MySQL, y haber creado la base de datos `concesionario` mostrada en la lectura anterior[2], así como sus tablas. No está de más haber agregado algunos registros en las tablas, tal como también se ilustró en la lectura anterior.

[1]El ejemplo completo se puede encontrar aquí: https://github.com/leonjaramillo/udea_ruta2_ciclo2/tree/main/main/java/co/edu/udea/udea_ruta2_ciclo2/db

[2]El script para la creación de dicha base de datos se puede encontrar aquí: https://github.com/leonjaramillo/udea_ruta2_ciclo2/blob/main/main/java/co/edu/udea/udea_ruta2_ciclo2/db/querys.sql

```java
package co.edu.udea.udea_ruta2_ciclo2.db;

import java.sql.*;

public class PruebaBaseDatos {

    public static void main(String[] args) {
        try {
            int documento;
            String nombres, apellidos, ciudad;
            //Se carga el driver correspondiente a MySQL
(versión 8)
            Class.forName("com.mysql.cj.jdbc.Driver");
            /* Se obtiene una conexión a la base de datos
usando el DriverManager
            Nos conectaremos a un servidor SQL en nuestra
máquina (localhost)
            y a una base de datos existente llamada
concesionario */
            String url = "jdbc:mysql://localhost/
concesionario";
            Connection conexion = DriverManager.
getConnection(url, "root", "");
            //Se crea una consulta, en este caso para
obtener todos los registros de la tabla vendedores
            String sentencia = "SELECT * FROM vendedores;";
            Statement consulta = conexion.
createStatement();
            //Se crea un ResultSet con los resultados de
la consulta y se itera sobre el mismo
            ResultSet resultados = consulta.
executeQuery(sentencia);
            while (resultados.next()) {
                documento = resultados.getInt("documento");
                nombres = resultados.getString("nombres");
                apellidos = resultados.
getString("apellidos");
                ciudad = resultados.getString("ciudad");
                System.out.println("Listado de
Vendedores");
                System.out.println("Documento: " +
documento +
                        "Nombres: " + nombres +
                        "Apellidos: " + apellidos +
                        "Ciudad" + ciudad);
}
```

```
            conexion.close();
        } catch (ClassNotFoundException e) {
            System.out.println("No fue posible cargar el
driver.");
        } catch (SQLException e) {
            System.out.println("Hubo un error al acceder a
la base de datos: " + e.getMessage());
        }

    }
}
```

Si nos fijamos detalladamente, podremos apreciar los siguientes pasos, comunes en muchos casos similares a la hora de acceder a una base de datos en MySQL desde Java usando JDBC:

1. Se registró el *driver* de MySQL en el `DriverManager` de Java, para que esto funcione es muy importante contar, como se mencionó más arriba, con el driver de MySQL configurado como dependencia dentro del proyecto.
2. Luego, se definió una cadena denominada `url`, a la cual se le asignó la cadena de conexión de la base de datos. La cadena de conexión sigue un formato estándar para MySQL y, al final, tiene dos valores importantes: el nombre o dirección IP del host o equipo donde está la base de datos (`localhost`, es decir, el equipo local), y el nombre de la base de datos ya existente (`concesionario`).
3. Después, se crea un objeto `conexion` del tipo `Connection`. Para esto se usa el método `getConnection` de la clase `DriverManager`, pasándole tres argumentos: la cadena de conexión definida arriba, el nombre de usuario (en este caso, `root`) y la contraseña (en este caso, vacía) usados para acceder a la base de datos. El nombre de usuario y la contraseña son aquellos que tenemos definidos para acceder al servidor de bases de datos desde que instalamos MySQL.

4. A continuación, definimos la consulta SQL que usaremos para recuperar los datos de la tabla `vendedores`. Esta será una consulta `SELECT`, que nos traerá todos los registros de dicha tabla. Para eso, usaremos un objeto del tipo `Statement`.
5. Luego, ejecutamos la consulta (con `executeQuery`), asignando sus resultados a un `ResultSet` del mismo nombre.
6. Un `ResultSet`, es un objeto que guarda los registros recuperados desde la base de datos mediante la consulta. Es aquí donde lo iteramos. Mediante un ciclo `while` que iterará mientras haya registros, obtenemos los valores de cada campo de cada uno de los registros de `resultados` y los imprimimos en pantalla.
7. Finalmente, cerramos la conexión, con el método `close`.

A continuación, podemos apreciar otro ejemplo basado en la misma base de datos. En este caso, insertamos un registro nuevo en la misma tabla.

```
package co.edu.udea.udea_ruta2_ciclo2.db;

import java.sql.*;

public class EjemploBaseDatosInsert {

    public static void main(String[] args) {
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            String url = "jdbc:mysql://localhost/
concesionario";
            Connection conexion = DriverManager.
getConnection(url, "root", "");
            String sentencia = "INSERT INTO vendedores
(documento,nombres,apellidos,ciudad) VALUES
('1379','Pepito','Pérez','Barranquilla');";
            Statement consulta = conexion.
createStatement();
            consulta.executeQuery(sentencia);
            conexion.close();
        } catch (ClassNotFoundException e) {
            System.out.println("No fue posible cargar el
driver.");
        } catch (SQLException e) {
            System.out.println("Hubo un error al acceder a
la base de datos: " + e.getMessage());
        }

    }
}
```

Notemos que en este caso no fue necesario asignar el resultado de la consulta a un `ResultSet`, dado que una consulta de tipo `INSERT` no devuelve registros. Por lo demás, este tipo de consultas se realiza de manera similar a las consultas `SELECT`. Las consultas para actualizar (`UPDATE`) y eliminar (`DELETE`) registros de una tabla se realizan de una forma similar.

Más Información

En internet existe mucha información sobre las funcionalidades y el uso de JDBC. Al buscarla, siempre es importante tener en cuenta la actualidad de la información, así como que el código fuente mostrado esté orientado al SGBD que vamos a usar. También, con el tiempo, es importante fijarnos en que el material a estudiar siga buenas prácticas y patrones de diseño usados en la industria. Un buen punto para empezar siempre será el curso oficial (en inglés): https://docs.oracle.com/javase/tutorial/jdbc/index.html