

Bootcamp Java Developer

Fase 1 - Java Analyst
Módulo 9

JDBC

Introducción

Java Database Connectivity (JDBC) es una **API de acceso a bases de datos estándar SQL** que proporciona un acceso uniforme a una gran variedad de bases de datos relacionales.

JDBC también proporciona una base común para la construcción de herramientas y utilidades de alto nivel. El paquete actual de JDK incluye JDBC.

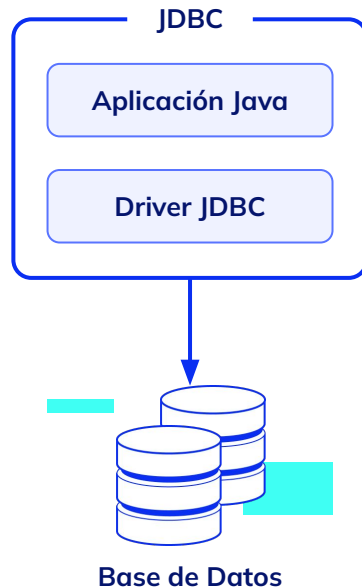
La API de JDBC se compone de dos paquetes:

- **java.sql:** paquete para acceder y procesar datos almacenados en una fuente de datos (generalmente una base de datos relacional).
- **javax.sql:** paquete que sirve para acceso y procesamiento de fuentes de datos del lado del servidor.



JDBC

Consiste en un **conjunto de clases e interfaces escritas en el lenguaje de programación Java**. JDBC suministra un **API estándar para los desarrolladores** y hace posible escribir aplicaciones de base de datos usando un API puro Java.

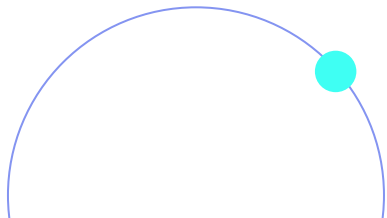


Librerías adicionales

Al trabajar con JDBC resulta **necesario agregar un *jar* al proyecto que contiene las clases necesarias que se utilizan para dialogar” con un DBMS**. Cada DBMS tiene su propio archivo *jar*. Recordemos que los proyectos están creados con un arquetipo de **Maven** y, a través del archivo ***pom.xml***, podemos gestionar las librerías que necesitamos.

Pasos a seguir

1. Nuestro proyecto trabaja con MariaDB o MySQL así que nos dirigimos al **repositorio de Maven** y buscamos el **Driver de MariaDB**: [MariaDB Java Client](#).
2. Seleccionamos la versión, copiamos la **dependencia** y la pegamos en el ***pom***, como podemos ver en el código del siguiente slide.



```
<!-- https://mvnrepository.com/artifact/org.mariadb.jdbc/mariadb-java-client -->  
<dependency>  
  <groupId>org.mariadb.jdbc</groupId>  
  <artifactId>mariadb-java-client</artifactId>  
  <version>2.7.2</version>  
</dependency>
```



Connection

Esta interfaz provee una **sesión de trabajo con una base de datos**. Sus métodos, aparte de permitir modificar y consultar **tablas**, permiten obtener información sobre su estructura:

Tipo	Método	Descripción
void	<code>commit()</code>	Confirma los cambios realizados provisionalmente por las transacciones.
void	<code>close()</code>	Libera la conexión a la base de datos.
Statement	<code>createStatement()</code>	Crea un objeto Statement para enviar sentencias SQL a la base de datos.
PreparedStatement	<code>prepareStatement(String sql)</code>	Crea un objeto PreparedStatement para enviar sentencias SQL parametrizadas a la base de datos.
void	<code>rollback()</code>	Deshace todos los cambios realizados en la transacción actual y libera los bloqueos de la base de datos.
void	<code>setAutoCommit(boolean autoCommit)</code>	Establece el modo de confirmación automática.
boolean	<code>isClosed()</code>	Recupera si la conexión se ha cerrado.

Crear la conexión

1. Para registrar el driver de la librería descargada lo primero que debemos hacer es **indicar la clase** que nos provee dicha librería.

```
String driver = "org.mariadb.jdbc.Driver";  
Class.forName(driver);
```

1. Una vez registrado el driver debemos **establecer la conexión a través de un método estático** que nos provee la clase denominada **DriverManager** **getConnection(String url, String user, String password)** y nos retorna un objeto de tipo **Connection**.

- La **url** espera el *driver*, el nombre de la PC o su IP, el puerto de conexión a la base de datos (por defecto en MariaDB y MySQL 3306) y el nombre de la base de datos.
- El **usuario** espera el nombre del usuario que se conecta a la base de datos, por defecto en MariaDB y MySQL es root.
- La **clave** espera por defecto la contraseña que posee el usuario, dependiendo del tipo de instalación puede estar vacía o una clave colocada al momento de instalar.

```
String url = "jdbc:mariadb://PC:PUERTO/BASE_DE_DATOS";  
String usuario = "USUARIO";  
String clave = "CLAVE";  
  
Connection conexion = null;  
conexion = DriverManager.getConnection(url, usuario, clave);
```

Ejecutar instrucciones SQL

La interfaz **Statement** provee una serie de **métodos que, al ser ejecutados, devuelven los resultados que producen las sentencias.**

Al crear la conexión, el objeto provee - como pudimos ver en el cuadro de métodos de **Connection** - el método **createStatement()** que nos devuelve un **Statement**.

El método **execute()** se utiliza para ejecutar sentencias SQL del tipo **INSERT**, **UPDATE** o **DELETE**.

Veamos la tabla de la próxima pantalla.



Tipo	Método	Descripción
boolean	<code>execute(String sql)</code>	Ejecuta la instrucción SQL dada, que puede devolver varios resultados.
ResultSet	<code>executeQuery(String sql)</code>	Ejecuta la instrucción SQL dada, que devuelve un solo Objeto ResultSet.

```
String sql = "INSERT, UPDATE, DELETE";  
Statement declaracionSQL = conexion.createStatement();  
declaracionSQL.execute(sql);
```

Insertar Registros

```
// CREAMOS LA CONEXION
conexion = DriverManager.getConnection(url, usuario, clave);

// CREAMOS NUESTRO QUERY SQL
String sql = "INSERT INTO autoFamiliar (patenteNumero, patenteActiva, marca, categoria, color, puestos) "
            + "VALUES ('ABC-001', 1, 'Audi', 'SEDAN', 'MARRON', 6);";

// CREAMOS NUESTRO OBJETO Statement
Statement declaracionSQL = conexion.createStatement();

// EJECUTAMOS EL SQL
declaracionSQL.execute(sql);

// CERRAMOS LA DECLARACION Y LA CONEXION
declaracionSQL.close();
conexion.close();
```

ResultSet

Para recuperar la información de una tabla o un origen de datos, Java proporciona la interfaz **ResultSet**. Esta interfaz provee varios métodos para obtener los datos de columna que corresponden a una fila. Todos ellos tienen el formato **get<Tipo>**, siendo <Tipo> un tipo de datos Java.

Ejemplos: `getInt`, `getLong`, `getString`, `getBoolean` y otros métodos.

Casi todos toman **un solo parámetro**, que es el índice que la columna tiene dentro del **ResultSet** o bien el nombre de la columna.

El método **executeQuery(String sql)** de la interfaz **Statement** devuelve el objeto.

Para poder **recuperar cada una de las filas** del objeto, si las hubiera, se utiliza frecuentemente el **bucle while con el método next() como condición**, que devuelve un boolean que indica si existe una fila a leer. Si existe, mueve el cursor una fila hacia adelante desde su posición actual.



Seleccionar Registros

```
// CREAMOS LA CONEXION
conexion = DriverManager.getConnection(url, usuario, clave);

// CREAMOS NUESTRO QUERY SQL
String sql = "SELECT patenteNumero, patenteActiva, marca, categoria, color, puestos FROM autoFamiliar;";

// CREAMOS NUESTRO OBJETO Statement
Statement declaracionSQL = conexion.createStatement();

// OBTENEMOS EL CONJUNTO DE RESULTADOS
ResultSet resultado = declaracionSQL.executeQuery(sql);
```

...



```
// RECORREMOS EL RESULTADO
while (resultado.next()) {
    System.out.println(resultado.getString("patenteNumero"));
    System.out.println(resultado.getBoolean("patenteActiva"));
    System.out.println(resultado.getString("marca"));
    System.out.println(resultado.getString("categoria"));
    System.out.println(resultado.getString("color"));
    System.out.println(resultado.getInt("puestos"));
    System.out.println();
}

// CERRAMOS LA DECLARACION Y LA CONEXION
declaracionSQL.close();
conexion.close();
```

Actualizar Registros

```
// CREAMOS LA CONEXION
conexion = DriverManager.getConnection(url, usuario, clave);

// CREAMOS NUESTRO QUERY SQL
String sql = "UPDATE autoFamiliar SET patenteActiva = 1 WHERE patenteActiva = 0;";

// CREAMOS NUESTRO OBJETO Statement
Statement declaracionSQL = conexion.createStatement();

// EJECUTAMOS EL SQL
declaracionSQL.execute(sql);

// CERRAMOS LA DECLARACION Y LA CONEXION
declaracionSQL.close();
conexion.close();
```


Borrar Registros

```
// CREAMOS LA CONEXION
conexion = DriverManager.getConnection(url, usuario, clave);

// CREAMOS NUESTRO QUERY SQL
String sql = "DELETE FROM autoFamiliar WHERE patenteActiva <> 0;";

// CREAMOS NUESTRO OBJETO Statement
Statement declaracionSQL = conexion.createStatement();

// EJECUTAMOS EL SQL
declaracionSQL.execute(sql);

// CERRAMOS LA DECLARACION Y LA CONEXION
declaracionSQL.close();
conexion.close();
```

**¡Sigamos
trabajando!**