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.

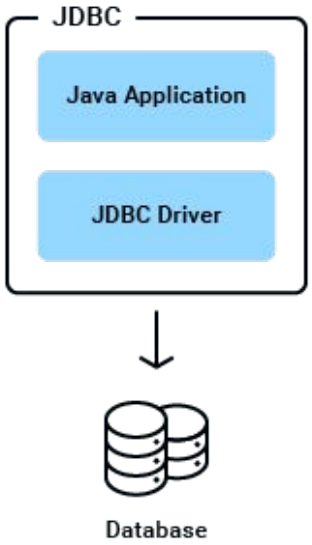
El paquete actual de JDK incluye JDBC.

La API de JDBC se compone de dos paquetes:

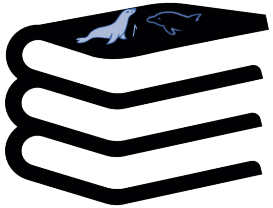
- **java.sql:** Paquete para acceder y procesar datos almacenados en una fuente de datos (generalmente una base de datos relacional).
- **javax.sql:** Paquete para acceso y procesamiento de fuentes de datos del lado del servidor.

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 nuestros proyectos están creados con un arquetipo de Maven y a través del archivo pom.xml podemos gestionar las librerías que necesitamos.

Nuestro proyecto trabajar con MariaDB o MySQL asi que nos dirigimos al repositorio de Maven y buscamos el Driver de MariaDB:

https://mvnrepository.com/artifact/org.mariadb.jdbc/mariadb-java-client

Seleccionamos la versión, copiamos la dependencia y la pegamos en el pom.

```
<!-- 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 modificarla y consultar sus tablas, también permiten obtener información sobre su estructura.

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

Crear la conexión

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);
```

Una vez registrado el driver debemos establecer la conexión a través de un método estático que nos provee la clase **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** nos provee una serie de métodos que al ser ejecutados nos devuelven los resultados que producen las sentencias ejecutadas.

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

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

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

```
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 nos proporciona la interfaz ResultSet.

Esta interfza nos provee de varios métodos para obtener los datos de columna correspondientes a un fila. Todos ellos tienen el formato get<Tipo>, siendo <Tipo> un tipo de datos Java. Algunos ejemplos de estos métodos son getInt, getLong, getString, getBoolean y etc.

Casi todos estos métodos 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** nos devuelve dicho objeto.

Para poder recuperar cada una de las filas del objeto si las hubiera utilizaremos por lo común el bucle while colocando como condición el método next() que devuelve un boolean indicando si existe una fila que leer, si existe dicha fila mueve el cursor una fila hacia adelante desde su posición actual.

Seleccionar Registros

```java
// 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();
```