### Ejecución de Sentencias SQL en Java con JDBC

Usamos la interfaz **Statement** para ejecutar sentencias SQL:

**SELECT** para recuperar datos.

**INSERT**, **UPDATE**, **DELETE** para modificar datos.

**Ejemplo**: Ejecutar una consulta

Statement stmt = con.createStatement();

ResultSet rs = stmt.executeQuery("SELECT \* FROM productos");

while (rs.next()) {

System.out.println(rs.getString("nombre\_producto"));

}

La **ejecución de sentencias SQL en Java con JDBC** es una de las tareas fundamentales al interactuar con bases de datos. JDBC proporciona las herramientas necesarias para ejecutar sentencias SQL como SELECT, INSERT, UPDATE y DELETE directamente desde un programa Java, utilizando objetos como Statement, PreparedStatement y CallableStatement.

### Tipos de Sentencias en JDBC:

1. **Statement**:
   * Es la clase más básica que se usa para ejecutar sentencias SQL simples y directas. Un Statement se utiliza cuando no es necesario pasar parámetros a la consulta SQL.
   * **Ejemplo**: Ejecutar una consulta SQL para obtener todos los productos de una tabla.

Statement stmt = con.createStatement();

ResultSet rs = stmt.executeQuery("SELECT \* FROM productos");

while (rs.next()) {

System.out.println(rs.getString("nombre\_producto"));

}

* + **Ventajas**: Sencillo de usar.
  + **Desventajas**: No es seguro frente a ataques de inyección SQL si los datos de usuario se incluyen directamente en la consulta.

1. **PreparedStatement**:
   * Esta clase es una mejora sobre Statement y permite ejecutar sentencias SQL precompiladas, lo que mejora la eficiencia cuando una misma consulta se ejecuta varias veces.
   * Lo más importante es que los **PreparedStatements** son seguros frente a **inyecciones SQL**, ya que los parámetros se pasan de forma segura y no directamente en la cadena SQL.

**Ejemplo**:

PreparedStatement pstmt = con.prepareStatement("SELECT \* FROM productos WHERE precio > ?");

pstmt.setDouble(1, 50.0); // Establece el parámetro

ResultSet rs = pstmt.executeQuery();

while (rs.next()) {

System.out.println(rs.getString("nombre\_producto"));

}

*Explicación del código:*

PreparedStatement: Se usa para crear una consulta SQL precompilada que permite incluir parámetros de forma segura. En lugar de concatenar valores directamente en el SQL (lo cual es inseguro y vulnerable a inyecciones SQL), PreparedStatement usa "marcadores de posición" (?), que luego serán reemplazados con valores.

con.prepareStatement(...): Llama al método prepareStatement sobre el objeto con (una conexión a la base de datos), generando un PreparedStatement basado en el SQL SELECT \* FROM productos WHERE precio > ?.

**Consulta SQL**: Selecciona todos los registros de la tabla productos donde el valor de la columna precio sea mayor a un valor que se establecerá después.

pstmt.setDouble(1, 50.0);: Asigna el valor 50.0 al primer marcador de posición (?) en la consulta.

**Primer parámetro (**1**)**: Especifica la posición del marcador de posición, en este caso, el primero (?).

**Segundo parámetro (**50.0**)**: Es el valor que será utilizado en la consulta, de tipo double.

Esta línea establece que se seleccionarán productos con un precio mayor a 50.0.

pstmt.executeQuery(): Ejecuta la consulta SQL en la base de datos y devuelve un ResultSet, que contiene los resultados de la consulta.

ResultSet rs: Almacena los resultados devueltos, permitiendo iterar sobre ellos y acceder a cada fila recuperada.

while (rs.next()): Itera sobre cada fila del ResultSet. rs.next() mueve el cursor a la siguiente fila y devuelve true si existe, o false si no hay más filas.

rs.getString("nombre\_producto"): Recupera el valor de la columna nombre\_producto como una cadena de texto para la fila actual en el ResultSet.

System.out.println(...): Imprime el nombre del producto en la consola.

* + **Ventajas**: Seguridad frente a inyección SQL y mejora de rendimiento con consultas repetidas.
  + **Desventajas**: Ligeramente más complejo de usar que Statement.

1. **CallableStatement**:
   * Se utiliza para ejecutar procedimientos almacenados en la base de datos, que son bloques de código SQL almacenados en el servidor de la base de datos.
   * Los **procedimientos almacenados** son útiles para realizar operaciones complejas en el servidor sin necesidad de pasar todo el código SQL desde la aplicación.

**Ejemplo**:

CallableStatement cstmt = con.prepareCall("{CALL obtener\_productos(?, ?)}");

cstmt.setInt(1, 50);

cstmt.registerOutParameter(2, Types.INTEGER);

cstmt.execute();

int resultado = cstmt.getInt(2);

* + **Ventajas**: Ideal para operaciones complejas y para reducir el tráfico de red.
  + **Desventajas**: Depende de que el procedimiento esté predefinido en la base de datos.

### ResultSet: Obteniendo Resultados de las Consultas

Cuando se ejecutan consultas SELECT con JDBC, el resultado se devuelve en un **ResultSet**, que es un conjunto de filas que puedes recorrer para obtener los datos.

* **Navegación por el ResultSet**: Utiliza el método next() para moverte de una fila a la siguiente, y métodos como getString(), getInt(), etc., para obtener los valores de las columnas.

ResultSet rs = stmt.executeQuery("SELECT \* FROM productos");

while (rs.next()) {

System.out.println(rs.getString("nombre\_producto"));

System.out.println(rs.getDouble("precio"));

}

* **Tipos de ResultSet**: Hay diferentes tipos de ResultSet, como:
  + **TYPE\_FORWARD\_ONLY** (por defecto): Solo permite avanzar una fila a la vez.
  + **TYPE\_SCROLL\_INSENSITIVE**: Permite moverse hacia adelante y hacia atrás en el ResultSet.
  + **TYPE\_SCROLL\_SENSITIVE**: Igual que el anterior, pero sensible a cambios realizados en la base de datos mientras se navega por el conjunto de resultados.

### Actualización de Datos con JDBC:

Las sentencias **INSERT**, **UPDATE** y **DELETE** también se ejecutan usando Statement o PreparedStatement, pero en lugar de devolver un ResultSet, se devuelve un valor entero que indica cuántas filas fueron afectadas.

PreparedStatement pstmt = con.prepareStatement("UPDATE productos SET precio = ? WHERE id = ?");

pstmt.setDouble(1, 99.99);

pstmt.setInt(2, 1);

int filasAfectadas = pstmt.executeUpdate();

System.out.println("Filas actualizadas: " + filasAfectadas);