### Ejercicio con Statement (Vulnerable a Inyección SQL)

**Objetivo del Ejercicio**: Implementar un sistema de inicio de sesión básico en Java utilizando JDBC, primero con Statement y luego con PreparedStatement. Comparar ambos métodos y comprender cómo PreparedStatement previene ataques de inyección SQL.

**Descripción del Problema**:

1. **Contexto**: Imagina que trabajas como desarrollador de una aplicación en Java que se conecta a una base de datos MySQL. Tu tarea es crear un sistema de inicio de sesión simple que valide las credenciales de los usuarios. La base de datos contiene una tabla usuarios con al menos dos campos: nombre (nombre de usuario) y contraseña.
2. **Parte 1: Implementación con** Statement:
   * Implementa un programa que solicite al usuario ingresar su nombre de usuario y contraseña.
   * Utiliza Statement para crear y ejecutar una consulta SQL que valide las credenciales ingresadas. Si el nombre de usuario y la contraseña coinciden con un registro en la tabla usuarios, muestra un mensaje de "Inicio de sesión exitoso". De lo contrario, muestra "Usuario o contraseña incorrectos".
   * Observa cómo este método es vulnerable a ataques de inyección SQL si se utilizan valores manipulados como entrada.
3. **Parte 2: Implementación con** PreparedStatement:
   * Modifica el programa anterior para utilizar PreparedStatement en lugar de Statement para ejecutar la consulta SQL.
   * Asegúrate de que los valores del nombre de usuario y la contraseña se establezcan mediante parámetros en PreparedStatement.
   * Realiza pruebas con las mismas entradas que usaste en la Parte 1 y observa cómo PreparedStatement evita la ejecución de consultas manipuladas y protege contra inyecciones SQL.
4. **Prueba y Comparación**:
   * **Para la Parte 1**, intenta iniciar sesión ingresando como nombre de usuario admin y como contraseña ' OR '1'='1.
   * **Para la Parte 2**, usa la misma entrada de admin como nombre de usuario y ' OR '1'='1 como contraseña. Observa cómo PreparedStatement protege contra inyecciones SQL.
5. **Discusión**:
   * Escribe una breve conclusión sobre las diferencias entre Statement y PreparedStatement, explicando por qué PreparedStatement es más seguro y cómo evita las inyecciones SQL.

**Datos de Ejemplo para la Tabla** usuarios:

Supón que tienes la siguiente tabla en tu base de datos:

| nombre | contraseña |
| --- | --- |
| admin | admin123 |
| user1 | password1 |
| user2 | password2 |

**Requisitos**:

* Usar JDBC para conectarse a una base de datos MySQL.
* Crear dos versiones del código: una con Statement y otra con PreparedStatement.
* Probar ambas versiones y documentar las diferencias.

**SOLUCIÓN**

**Código con** Statement **(Vulnerable)**:

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.ResultSet;

import java.sql.Statement;

import java.util.Scanner;

public class LoginVulnerable {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Introduce tu nombre de usuario: ");

String nombreUsuario = scanner.nextLine();

System.out.print("Introduce tu contraseña: ");

String contraseña = scanner.nextLine();

String url = "jdbc:mysql://localhost:3306/tu\_base\_de\_datos";

String usuario = "root";

String contraseñaBD = "tu\_contraseña";

try (Connection con = DriverManager.getConnection(url, usuario, contraseñaBD);

Statement stmt = con.createStatement()) {

// Consulta vulnerable a inyección SQL

String sql = "SELECT \* FROM usuarios WHERE nombre = '" + nombreUsuario + "' AND contraseña = '" + contraseña + "'";

ResultSet rs = stmt.executeQuery(sql);

if (rs.next()) {

System.out.println("Inicio de sesión exitoso.");

} else {

System.out.println("Usuario o contraseña incorrectos.");

}

} catch (Exception e) {

e.printStackTrace();

}

}

}

**Inyección SQL**: Si el usuario ingresa como nombreUsuario el valor admin y como contraseña el valor ' OR '1'='1, el código SQL resultante sería:

SELECT \* FROM usuarios WHERE nombre = 'admin' AND contraseña = '' OR '1'='1';

Aquí, OR '1'='1' siempre es verdadero, por lo que la consulta devuelve todos los registros y permite el acceso sin una contraseña válida.

### Ejercicio con PreparedStatement (Protegido contra Inyección SQL)

**Descripción**: Este código realiza la misma tarea, pero ahora usa PreparedStatement para prevenir la inyección SQL. Los datos del usuario se pasan como parámetros, de forma que el motor de la base de datos los trata de forma segura, evitando interpretarlos como parte de la consulta.

**Código con** PreparedStatement **(Seguro)**:

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.PreparedStatement;

import java.sql.ResultSet;

import java.util.Scanner;

public class LoginSeguro {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Introduce tu nombre de usuario: ");

String nombreUsuario = scanner.nextLine();

System.out.print("Introduce tu contraseña: ");

String contraseña = scanner.nextLine();

String url = "jdbc:mysql://localhost:3306/tu\_base\_de\_datos";

String usuario = "root";

String contraseñaBD = "tu\_contraseña";

try (Connection con = DriverManager.getConnection(url, usuario, contraseñaBD)) {

// Consulta segura con PreparedStatement

String sql = "SELECT \* FROM usuarios WHERE nombre = ? AND contraseña = ?";

PreparedStatement pstmt = con.prepareStatement(sql);

pstmt.setString(1, nombreUsuario); // Primer parámetro

pstmt.setString(2, contraseña); // Segundo parámetro

ResultSet rs = pstmt.executeQuery();

if (rs.next()) {

System.out.println("Inicio de sesión exitoso.");

} else {

System.out.println("Usuario o contraseña incorrectos.");

}

} catch (Exception e) {

e.printStackTrace();

}

}

}

**Explicación de** PreparedStatement:

* Con PreparedStatement, el SQL utiliza marcadores (?) para los valores del nombre de usuario y la contraseña.
* pstmt.setString(1, nombreUsuario) y pstmt.setString(2, contraseña): Establecen los valores de manera segura y no se interpretan como código SQL.
* Esto evita que cualquier inyección SQL afecte la consulta, ya que los valores se procesan independientemente.

**Ventaja**:

* El código es seguro contra inyecciones SQL. La base de datos trata los valores de nombreUsuario y contraseña solo como datos, sin mezclarlos con la lógica de la consulta.