2. Sentencias preparadas

En los ejemplos vistos hasta ahora, las sentencias de SQL eran fijas, estaban en cadenas de caracteres constantes. En una aplicación real suele ser necesario ejecutar sentencias de SQL en las que intervengan variables, bien porque dependan de valores introducidos desde la interfaz de usuario de una aplicación (por ejemplo, un formulario de consulta para clientes con diversos criterios de búsqueda, tales como DNI, nombre, etc), o bien porque dependan de un dato obtenido desde una fuente de datos (como un fichero o una consulta en SQL).

Se podría crear la sentencia en un String y asignarle una expresión en la que intervengan variables. Por ejemplo:

```
String sql = "SELECT * FROM clientes WHERE dni='" + dni + "'";
```

Pero este planteamiento presenta una serie de problemas que obligan a descartarlo:

1. **Seguridad**: una sentencia SQL construida en tiempo de ejecución utilizando variables está expuesta a ataques mediante técnicas de inyección SQL (*SQL Injection*). Estas son técnicas para lograr que código SQL introducido como parte del contenido de estas variables (como dni en el ejemplo anterior) se ejecute. Esto permite al atacante consultar datos e incluso modificarlos y borrarlos. El riesgo es especialmente grande cuando los valores se introducen desde una interfaz de usuario. Estos ataques se podrían evitar verificando el contenido de las variables, pero esto resulta complejo y no elimina completamente el riesgo.
2. **Rendimiento**: una consulta que se envía al SGBD se compila antes de ejecutarse. Es decir, se analiza y se crea un plan de ejecución para ella. Si entre una consulta y otra solo cambia el valor de algunas variables, se compilará cada una de las consultas para ejecutarse siempre de la misma forma, lo que resulta poco eficiente.

Las sentencias preparadas permiten evitar estos problemas. Una sentencia preparada permite incluir marcadores (*placeholders*) en determinados lugares de la sentencia donde van valores que se proporcionarán en el momento de ejecutar la sentencia.

Una sentencia preparada se le proporciona al servidor de base de datos solo una vez, y este la prepara o precompila. A partir de ahí, solo hay que pasar un valor para cada *placeholder* cada vez que se ejecuta la consulta.

2.1. La interfaz PreparedStatement

Para hacer consultas con sentencias preparadas se utiliza PreparedStatement. Se obtiene un PreparedStatement con el método prepareStatement() de la interfaz Connection. A este método se le pasa como parámetro la sentencia SQL y se utiliza el carácter "?" como *placeholder*. Por ejemplo, vamos a crear una sentencia preparada con dos parámetros de entrada:

```
Connection conexion = DriverManager.getConnection(url, "usuario", "contraseña");
PreparedStatement sentencia = conexion.prepareStatement("UPDATE tabla SET campo1=? WHERE campo2=?");
```

Antes de poder ejecutar un objeto PreparedStatement, se le debe asignar un valor para cada uno de sus parámetros. Esto se realiza con la llamada a los métodos setXXX, donde XXX es el tipo de dato apropiado para el parámetro.

Si el valor que queremos sustituir por un "?" es un int de Java, podremos llamar al método setInt(). Si el valor fuese un String, podemos llamar al método setString(). Y así con los demás tipos de datos.

La sintaxis de estos métodos es: `setXXX(posición, valor)`

El primer argumento es la posición del parámetro al que se le va a asignar el valor, mientras que el segundo argumento es el valor a asignar.

Por ejemplo, en el siguiente código creamos un objeto PreparedStatement y le asignamos al primero de sus parámetros un valor de tipo String y al segundo un valor de tipo int:

```
PreparedStatement sentencia = conexion.prepareStatement("UPDATE tabla SET nombre=? WHERE edad=?");
sentencia.setString(1, "Pepe");
sentencia.setInt(2, 23);
```

En este caso, el conectador enviará a la base de datos la cadena "Pepe" como un VARCHAR y el valor 23 como un INT. El programador debe asegurarse de que el tipo de dato Java de cada parámetro sea convertido a un tipo de dato compatible de la base de datos. En general, tendremos un método setXXX para cada tipo Java.

Para ejecutar la sentencia, disponemos de los mismos métodos que con Statement:

- executeQuery(): se utiliza con SELECT.
- executeUpdate(): se utiliza con cualquier otra sentencia (UPDATE, DELETE, INSERT…).
- execute(): se puede utilizar con cualquiera.

La única diferencia es que estos métodos no reciben ningún argumento, ya que la sentencia SQL correspondiente se indica con el método prepareStatement().

A continuación, se muestra un programa de ejemplo en el que se insertan en una tabla CLIENTES1 los datos de tres clientes, utilizando una sentencia preparada:

```
Connection c = DriverManager.getConnection(urlConnection, user,
    pwd);
PreparedStatement sInsert = c.prepareStatement("INSERT INTO
    CLIENTES1(DNI,APELLIDOS,CP) VALUES (?,?,?)") {
sInsert.setString(1, "78901234X");
sInsert.setString(2, "NADALES");
sInsert.setInt(3, 44126);
sInsert.executeUpdate();
```

EJERCICIO PROPUESTO

1. Utilizando la base de datos de *empleados*, crea una clase llamada *ConsultaNombres* que devuelva los nombres de los empleados que empiezan por una letra determinada. Esta letra será introducida por el usuario.
2. Utilizando la misma base de datos, crea una clase llamada *BorradoEmpleados* que permita borrar un empleado con un número determinado. Este número será introducido por el usuario.