educación IT
CODE YOUR FUTURE

Java Standard Web Programming

Módulo 6

Declaración Preparada

Introducción

Los gestores de base de datos relacionales (como MySQL, PostgreSQL, SQL Server, y otros) proporcionan mecanismos para optimizar el acceso y ejecución de sentencias.

Uno de estos mecanismos es la ***declaración preparada***; forma con la que le **indicamos a la base de datos que vamos a utilizar una sentencia y que la guarde y la "precompile" con las condiciones necesarias para poder utilizarla**.

Muchas veces, necesitaremos **ejecutar**, más de una vez, **la misma sentencia pero con valores diferentes**. Al hacerlo con una declaración preparada, evitamos que el gestor evalúe cada vez si lo que le enviamos tiene la sintaxis correcta, si los campos y tablas existen y la conversión al tipo de dato es correcto.

Ejemplo:

```
-- USAR BASE DE DATOS
USE autosEducacionIT;

-- INSERTAR ESPECIFICANDO LOS CAMPOS PREPARANDO LA
PREPARE INSERTAR FROM "INSERT INTO autosEducacionIT.autoFamiliar
(patenteNumero, patenteActiva, marca, categoria, color, puestos)
VALUES  (?, ?, ?, ?, ?, ?);";

-- SETEAMOS LAS VARIABLES
SET @patenteNumero  = 'ABC-001';
SET @patenteActiva = 1;
SET @marca = 'Audi';
SET @categoria = 'SEDAN';
SET @Color = 'ROJO';
SET @puestos =  6;
```

...

```
...

-- EJECUTAR LA CONSULTA PREPARADA
EXECUTE INSERTAR USING @patenteNumero, @patenteActiva, @marca, @categoria, @Color, @puestos;

-- SETEAMOS LAS VARIABLES
SET @patenteNumero  = 'ZXM-901';
SET @patenteActiva = 1;
SET @marca = 'Ford';
SET @categoria = 'COMPACTO';
SET @Color = 'NEGRO';
SET @puestos =  4;

-- EJECUTAR LA CONSULTA PREPARADA
EXECUTE INSERTAR USING @patenteNumero, @patenteActiva, @marca, @categoria, @Color, @puestos;
```

Ejecutar declaraciones preparadas SQL

La interfaz **`PreparedStatement`**, que hereda **`Statement`**, proporciona un objeto que representa una **instrucción SQL precompilada.** Además, proporciona una serie de métodos que, al ser ejecutados, devuelven los resultados que producen dichas sentencias.

Al crear la conexión, el objeto provee - como pudimos ver en el cuadro de métodos de **`Connection`** - el método denominado **`prepareStatement(String sql)`** que nos devuelve un **`PreparedStatement`**.

Tipo	Método	Descripción
`boolean`	`execute(String sql)`	Ejecuta la instrucción SQL dada, que puede devolver varios resultados.
`ResultSet`	`executeQuery()`	Ejecuta la instrucción SQL dada, que devuelve un solo Objeto `ResultSet`.
`Int`	`executeUpdate()`	Ejecuta la instrucción SQL dada, que debe ser de tipo DML y devuelve uno (1) si se realizó correctamente o cero (0).

Parámetros

La interfaz provee además varios **métodos `setter`** para establecer los valores de los parámetros: **`setTipoDato(int indiceParametro, TipoDato valorDato)`**.

Estos parámetros están representados por el signo de interrogación "**`?`**", en el `String` que contiene el SQL, al usar los métodos `setter` debemos indicar el índice del parámetro a sustituir comenzando desde el número 1.

¿Qué ganamos con esto además de la optimización del proceso? Que **al sustituir un valor no se arme de manera dinámica el SQL y así se evita la inyección SQL**.

Veamos un ejemplo
en la próxima pantalla.

```
String sql = "INSERT ?, UPDATE ?, DELETE ?";

PreparedStatement declaracionPreparadaSQL = conexion.prepareStatement(sql);

declaracionPreparadaSQL.setTipo(1, valor);

declaracionPreparadaSQL.execute();
```

Inyección SQL

Introducción

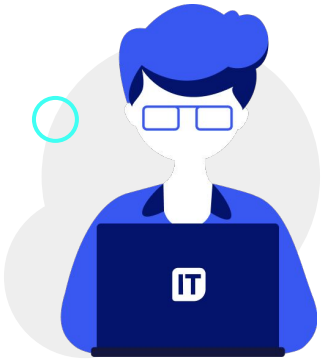
Al comenzar en el mundo del desarrollo y dar los primeros pasos en la persistencia de información con una base de datos relacional, ejecutamos sentencias SQL concatenando los valores y así poder enviar la información pertinente al gestor de base de datos. Esto sucede por lo general cuando es necesario generar sentencias dinámicas y enviar un valor en el SQL que se le pide al usuario, por ejemplo: correo, contraseña, filtros y/o datos en general.

Inyección SQL

El **problema es que al concatenar la información sin validar, puede venir con una Inyección SQL**, que no es más que la técnica de **introducir código malicioso en las declaraciones SQL**. Esto puede perjudicar mucho a nuestros datos.

Ejemplo:

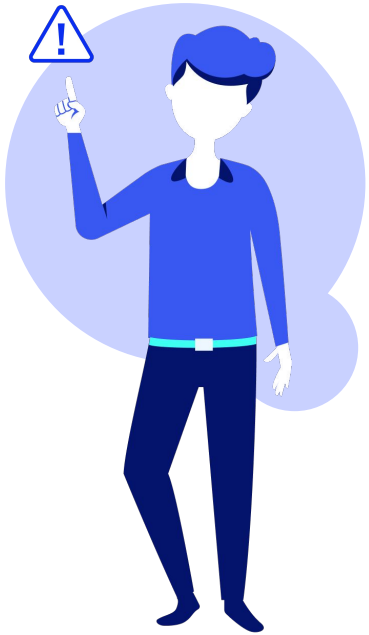
```
String filtroCampo1 = "VALOR";
String seleccionar = " SELECT Campo1, Campo2 FROM TABLA WHERE Campo1 = '" + filtroCampo1 + "';";
```



Inyección SQL

A pesar de que parece una tontería que nuestro código sea así de vulnerable, la **inyección SQL** sigue siendo, según la fundación OWASP, **el primero de los riesgos de seguridad en sitios WEB y aplicaciones**.

Aunque existen otros tipos de vulnerabilidades que tienen que ver con la inyección de código, en este módulo, aprenderemos a evitar las de tipo SQL a través de herramientas que ofrecen los gestores de base de datos y Java.



educación IT
CODE YOUR FUTURE

¡Sigamos trabajando!