Tutoriales

- Java
- C/C++ de Linux
- Metodologías y diseño orientado a objetos
- CSS

Enlaces

- Diario de Programación
- Chuwiki
- Micro entradas
- Foro de Java y C++
- Mis proyectos
- Pasatiempos

Uso de PreparedStatement con Java y MySQL

Cuando trabajamos con una base de datos es posible que haya sentencias *SQL* que tengamos que ejecutar varias veces durante la sesión, aunque sea con distintos parámetros. Por ejemplo, durante una sesión con base de datos podemos querer insertar varios registros en una tabla. Cada vez los datos que insertamos serán distintos, pero la sentencia *SQL* será la misma: Un *INSERT* sobre determinada tabla que será simpre igual, salvo los valores concretos que queramos insertar.

Casi todas las bases de datos tienen previsto un mecanismo para que en estos casos la ejecución de esas sentencias repetidas sea más rápida. Si tenemos una tabla person con un id, una edad, un nombre, un apellido y hacemos, por ejemplo, varios *INSERT* así

```
mysql> INSERT INTO person VALUES (null, 23, 'Pedro', 'Perez');
mysql> INSERT INTO person VALUES (null, 33, 'Rodrigo', 'Rodriquez');
```

en cada caso la base de datos deberá analizar la sentencia *SQL*, comprobar que es correcta, convertir los datos al tipo adecuado (por ejemplo, los enteros a int) y ejecutar la sentencia.

El mecanismo que preven las bases de datos para hacer más eficiente este proceso es que le indiquemos, previamente, el tipo de sentencia que vamos a usar, de forma que la base de datos la "*precompila*" y la guarda en condiciones de ser ejecutada inmediatamente, sin necesidad de analizarla en cada caso. Esto es lo que se conoce como una *prepared statement*. En el caso de *mysql*, se haría de esta forma

```
mysql> PREPARE insertar FROM "INSERT INTO person VALUES (null, ?, ?, ?)";
mysql> SET @edad=23;
mysql> SET @nombre='Pedro';
mysql> SET @apellido='Perez';
```

```
mysql> EXECUTE insertar USING @edad,@nombre,@apellido
mysql> SET @edad=33;
mysql> SET @nombre='Rodrigo';
mysql> SET @apellido='Rodriguez';
mysql> EXECUTE insertar USING @edad,@nombre,@apellido;
mysql> DEALLOCATE PREPARE insertar;
```

donde hemos preparado una *prepared statement* de nombre *insertar* con la *SQL* del *INSERT*, en la que hemos reemplazado los valores concretos por interrogantes. Fíjate que no hemos puesto comillas entre los interrogantes. Hemos hecho dos inserciones dando valores a unas variables *@edad*, *@nombre* y *@apellido*, que son las que se usarán en el *EXCECUTE* de la *prepared statement*. Una vez finalizadas las inserciones, avisamos a la base de datos que no vamos a usar más esta *prepared statement* con un *DEALLOCATE*.

La ejecución de los *insert* realizados de esta manera es, teóricamente, más eficiente que los realizados de la forma tradicional, escribiendo el *INSERT* directamente. Existe otra ventaja adicional en el tema de seguridad, pero eso afecta más al programa *java*. Puedes verla en el enlace.

PreparedStatement desde java. Establecer la conexión.

Si la base de datos soporta *prepared statement* y el *driver*/*conector* que usemos para hablar con esa base de datos desde *java* los soporta también, entonces podemos usar los *prepared statement* desde *java*.

Si la base de datos no los soporta o el *driver*/*conector* no los soporta, podemos desde *java* hacer el código igualmente. A la hora de codificar podemos usar los *PreparedStatement* independientemente de que la base de datos y/o el conector los soporten o no, pero no conseguiremos los beneficios deseados. Aunque usemos *PreparedStatement* desde java, por debajo acabarán traduciéndose a *statement* normal. Si quieres asegurarte de conseguir todos los beneficios, debes verificar si tu base de datos y el *driver* de conexión con ella soportan los *prepared statement*. En el caso de *MySQL*, las versiones modernas lo hacen.

Lo primero de todo es establecer la conexión. En el caso de *MySQL* debemos además poner un pequeño parámetro adicional para configurar el driver para que use las *prepared statement* en el servidor. Este parámetro es *useServerPrepStmts=true*, por lo que la cadena de conexión completa sería

```
try {
   Class.forName("com.mysql.jdbc.Driver");

   // Es necesario useServerPrepStmts=true para que los PreparedStatement
   // se hagan en el servidor de bd. Si no lo ponemos, funciona todo
   // igual, pero los PreparedStatement se convertirán internamente a
   // Statements.
   Connection conexion = DriverManager.getConnection(
      "jdbc:mysql://servidor/basedatos?useServerPrepStmts=true",
      "usuario", "password");
   ...
} catch (Exception e) {
   e.printStackTrace();
}
```

Crear el PreparedStatement

Una vez establecida la conexión, podemos crear el *PreparedStatement* llamando al método *prepareStatement()* de la *Connection*. Puesto que los *PreparedStatement* son importantes cuando queremos utilizarlos varias veces para ganar en eficiencia, es importante guardar este *PreparedStatement* en algún sitio al que podamos acceder cuando lo necesitemos. Si cada vez que vamos a usarlo creamos un *PreparedStatement* nuevo, tampoco conseguiremos la mejora de eficiencia. Un buen sitio para guardar este *PreparedStatement* puede ser un atributo de la clase.

```
public class UnaClase {
   // Aqui guardamos un unico PreparedStatement para insertar
   PreparedStatement psInsertar = null;
   ...
   public void unMetodoDeInsertar () {
   try {
      // Creamos el PreparedStatement si no estaba ya creado.
      if (null == psInsertar) {
```

```
        psInsertar = conexion.prepareStatement(
           "insert into person values (null,?,?,?)");
      ...
   } catch (SQLException e) {
      e.printStackTrace();
   }
```

Usar el PreparedStatement: Una pequeña ventaja en la seguridad

Una vez creado y guardado a buen recaudo, podemos usarlo siempre que nos haga falta. Para ello, primero debemos darle valor a los parámetros que dejamos como interrogantes. Usaremos los método *set()* de que dispone *PreparedStatement*, teniendo en cuenta que el primer interrogante que aparece en la *SQL* es el parámetro 1, el segundo el 2, etc. La inserción quedaría entonces

```
psInsertar.setInt(1, 23); // La edad, el primer interrogante, es un entero.
psInsertar.setString(2, "Pedro"); // El String nombre es el segundo interrogante
psInsertar.setString(3, "Perez"); // Y el tercer interrogante, un String apellido.
psInsertar.exequteUpdate(); // Se ejecuta la inserción.
```

Precisamente por esta forma de meter los datos, conseguimos una pequeña ventaja en el tema de seguridad. Imagínate que el nombre viene de un campo de texto que el usuario rellena. Si el usuario pone, por ejemplo, un nombre como "*0'Donnell*", con una comilla simple entre medias, nos estropearía una *Statement* normal que montásemos a base de concatenar cadenas

```
String sql = "insert into person values (null, " +
   edad + ",'" + nombre + "','" + apellido + "')";
```

Hemos ido sumando cadenas y hemos metido la variable *nombre* entre comillas simples, pero como nombre a su vez tiene comillas simples, el resultado de todo esto sería un *String* así

```
insert into person values (null, 23, 'O'Donnell', 'Perez')
```

que dará un error al usarlo, porque la comilla de *O'Donnell* se interpreta con fin de la cadena nombre y detrás debería ir, en una *SQL* correctamente formada, una coma para separar el campo apellido, y no una D mayúscula. Es más, un usuario con concimientos y mal intencionado, podría incluso conseguir que se ejecutase un trozo de *SQL* malicioso.

Si usamos *Statement* normal y componemos nosotros las *SQL*, es nuestra responsabilidad revisar las cadenas que nos llegan desde el usuario y "*escapar*" anteponiendo un \ en todos aquellos caracteres susceptibles de estropearnos la sentencia *SQL*.

Esto no pasa, sin embargo, con los *PreparedStatement*. Al crear la *PreparedStatement*, ya indicamos cómo es la *SQL* y qué campos tiene. Al meter dichos campos con métodos específicos, *setInt()*, *setString()*, etc, la *PreparedStatement* ya sabe cuales son exactamente los valores de los campos y cual exactamente la *SQL*, por lo que el nombre "*O'Donnell*" nos nos causaría ningún problema en la inserción.

Obtener las claves recién creadas

Al hacer un *insert*, si tenemos así configurada la tabla de base de datos, es posible que la clave primaria de la tabla se autogenere al hacer el *insert*. Con *PreparedStatement* podemos obtener esa clave recién generada al hacer el *insert*. Para ello, al crear la *PreparedStatement*, debemo dar un parámetro más indicando que tenemos interés en recuperar dicha clave.

```
PreparedStatement ps = conexion.prepareStatement(
   "insert into person values (null,?,?,?)",
   PreparedStatement.RETURN_GENERATED_KEYS);
```

Listo.. Una vez que hagamos un *insert*, podremos obtener la clave generada. En este ejemplo, la clave es ese campo *null* que hemos puesto en el *insert*. Al ser *null* y ser el campo en base de datos una clave autogenerada, *MySQL* generará un valor distinto cada vez. Para obtener ese valor

```
ps.setInt(1, 22);
ps.setString(2, "Juan");
ps.setString(3, "Perez");
ps.executeUpdate();

// Se obtiene la clave generada
```

```
ResultSet rs = ps.getGeneratedKeys();
while (rs.next()) {
   int claveGenerada = rs.getInt(1);
   System.out.println("Clave generada = " + claveGenerada);
}
```

Con el método *getGeneratedKeys()* obtenemos un *ResultSet* con las claves generadas. Puesto que sólo hemos hecho una inserción, ese *ResultSet* sólo tendrá una fila (el bucle while sólo se ejecutará una vez). Como la clave es un simple entero, estará en la columna 1 del *ResultSet* (*rs.getInt(1)*).

Ejemplo completo

Puedes ver el ejemplo completo de este tutorial en EjemploPreparedStatement.java

Estadísticas y comentarios

Numero de visitas desde el 4 Feb 2007:

- Esta pagina este mes: 299
- Total de esta pagina: 142533
- Total del sitio: 19099217