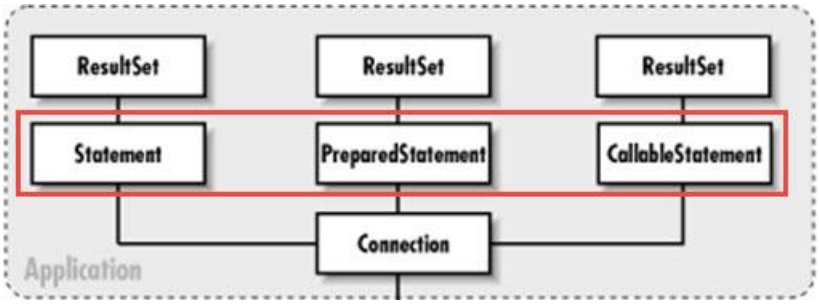
Manejo de conectores (3ª Parte)

Ya sabemos cómo establecer, desde un programa Java, la conexión con una base de datos mediante JDBC y obtener información sobre sus metadatos.

Ahora continuaremos estudiando los componentes JDBC que permiten ejecutar todo tipo de sentencia SQL.

1. Ejecución de sentencias SQL

Ahora pasaremos a estudiar los componentes JDBC que permiten ejecutar todo tipo de sentencia SQL. Dependiendo del tipo de instrucción a ejecutar utilizaremos un interface u otro y utilizaremos un método u otro para ejecutarla.



Para crear un objeto basado en una de estas interfaces usaremos el método correspondiente del objeto Connection.

- **Statement** se usa para ejecutar una sentencia SQL sin información dinámica (parámetros) contra la BD.

- **PreparedStatement** hereda de java.sql.Statement. Se diferencia de ella en que:
  - La sentencia SQL es compilada por el SGBD previamente.
  - La sentencia puede contener variables marcadas con ? (parámetros).

- **CallableStatement** hereda de java.sql.PreparedStatement, y sirve para ejecutar procedimientos almacenados en la BD.

1.1. ResultSet

Representa un conjunto de filas con sus columnas, en el caso de utilizarlos para almacenar el resultado de la ejecución de un Statement con la instrucción SELECT contendrá todas las filas resultantes de la SELECT.

Implementa métodos para:

- Acceder a las filas que componen el resultado.
- Acceder al valor de cada columna de la fila seleccionada.

Existen muchos métodos, nosotros veremos algunos de ellos:

El método para acceder a la fila siguiente del resultado es:

```
public boolean next() throws SQLException;
```

El método para acceder a la fila anterior del resultado es:

```
public boolean previous() throws SQLException;
```

El método a usar para acceder al valor de una columna, dependerá del tipo

de dato de la columna (xxxx), siendo 1 el valor del índice para la primera columna.

```
public xxxx getXXXX(int column) throws SQLException;
```

En la siguiente tabla están representados los métodos utilizables para cada tipo SQL.

Nota: una x indica una solución legal mientras que una X indica la solución recomendada.

getObject	getURL	getRef	getArray	getBlob	getClob	getCharacterStream	getBinaryStream	getAsciiStream	getTimestamp	getTime	getDate	getBytes	getString	getBoolean	getBigDecimal	getDouble	getFloat	getLong	getInt	getShort	getByte	
x													x	x	x	x	x	x	x	x	X	TINYINT
x													x	x	x	x	x	x	x	X	x	SMALLINT
x													x	x	x	x	x	x	X	x	x	INTEGER
x													x	x	x	x	x	X	x	x	x	BIGINT
x													x	x	x	x	X	x	x	x	x	REAL
x													x	x	x	X	x	x	x	x	x	FLOAT
x													x	x	x	X	x	x	x	x	x	DOUBLE
x													x	x	X	x	x	x	x	x	x	DECIMAL
x													x	x	X	x	x	x	x	x	x	NUMERIC
x													x	X	x	x	x	x	x	x	x	BIT
													x	X		x	x	x	x	x	x	BOOLEAN
x						x		x	x	x	x		X	x	x	x	x	x	x	x	x	CHAR
x						x		x	x	x	x		X	x	x	x	x	x	x	x	x	VARCHAR
x						X		X	x	x	x		x	x	x	x	x	x	x	x	x	LONGVARCHAR
x						x	x	x				X	x									BINARY
x						x	x	x				X	x									VARBINARY
x						x	X	x				x	x									LONGVARBINARY
x									x		X		x									DATE
x									x	X			x									TIME
x									X	x	x		x									TIMESTAMP
x					X																	CLOB
x				X																		BLOB
x			X																			ARRAY
x		X																				REF
x	X												x									DATALINK
X																						STRUCT
X																						JAVA_OBJECT

El ResultSet actua como un cursor y cuando se crea el cursor está posicionado antes de la primera fila, por lo que habrá que ejecutar un primer next() para acceder a la primera fila.

Se suele implementar un bucle de tipo while para recorrer y tratar todo el ResultSet.

Los ResultSets se cierran mediante el método:

```
public boolean close() throws SQLException;
```

Nota: el ResultSet se cierra automáticamente al cerrar el Statement que lo creó. No obstante, no está demás cerrarlo.

1.2. Statement

Como hemos dicho Statement se utiliza para ejecutar una sentencia SQL sin parámetros. Siempre lleva asociada una conexión que sirvió para crearlo.

Los objetos basados en esta interface se crean con el siguiente método de la clase java.sql.Connection:

```
public Statement createStatement() throws SQLException;
```

Ejemplo: Statement sentencia = miconexion.createStatement("SELECT * FROM T1);

Se cierran mediante el método:

```
public void close() throws java.sql.SQLException;
```

El método para ejecutarlo, depende del tipo de sentencia SQL que contenga.

Sentencia SELECT:

- Se usa el método: **executeQuery**(String sql);
- Devuelve una instancia de java.sql.**ResultSet**

Sentencia INSERT, UPDATE y DELETE:

- Se usa el método: **executeUpdate**(String sql);
- Devuelve un **int** con el número de filas afectadas.

Sentencia DDL:

- Se usa el método: **executeUpdate**(String sql);
- Devuelve un int que siempre vale **0**.

Ejemplos:

Sentencia SELECT sin parámetros:

```
Connection miconexion = DriverManager.getConnection("jdbc:sqlite:C:/Pruebas/sqlite/ejemplo1.db");
// Preparamos la consulta
Statement sentencia= miconexion.createStatement();
// Cargamos el resultado
ResultSet resultado= sentencia.executeQuery("SELECT loc, dept_no, dnombre FROM departamentos");
// Recorremos el resultado para visualizar cada fila
// Se hace un bucle mientras haya registros, se visualiza cada uno
while(resultado.next())
{
    // Con getTipodato(n) n indica el nº de la columna empezando por 1 y tipo dato el tipo devuelto
    System.out.println(resultado.getInt(2)+ " " + resultado.getString(1)+ " "+ resultado.getString(3));
    System.out.println(resultado.getString("dnombre")+ " " + resultado.getString("LOC")+ " "+ resultado.getString("dept_no"
    System.out.println(resultado.getString(1)+ " " + resultado.getString(2)+ " "+ resultado.getString(3));
}
// Liberamos recursos
resultado.close();
sentencia.close();
```

Sentencia INSERT sin parámetros:

```
Class.forName("com.mysql.jdbc.Driver");
// Establecemos la conexión con la base de datos
Connection miconexion = DriverManager.getConnection("jdbc:mysql://localhost/Ejemplo", "root", "root");
/ Recuperamos los argumentos del main()
System.out.println(args.length);
String dep=args[0];
String dnombre=args[1];
String loc=args[2];
// construimos la orden INSERT
String sql= "INSERT INTO departamentos VALUES (" + dep + ", '" + dnombre + "', '" + loc + "')";
// visualizamos la sql para comprobar que se ha construido bien
System.out.println(sql);
// Creamos la instruccion
Statement sentencia= miconexion.createStatement();
/ La ejecutamos recogiendo el nº de filas afectadas
int filas=sentencia.executeUpdate(sql);
/ Visualizamos el nº de filas afectadas
System.out.println("Nº de filas afectadas: " + filas);
// Liberamos recursos
sentencia.close();
miconexion.close();
```

Sentencia DDL sin parámetros:

```
Class.forName("com.mysql.jdbc.Driver");
// Establecemos la conexión con la base de datos
Connection miconexion = DriverManager.getConnection("jdbc:mysql://localhost/Ejemplo", "root", "root");
// construimos la orden CREATE VIEW
String sql= "CREATE OR REPLACE VIEW totales AS SELECT d.dept_no AS dep, dnombre, COUNT(emp_no) AS Empleac
          " FROM departamentos d LEFT JOIN empleados e ON d.dept_no = e.dept_no " +
          " GROUP BY d.dept_no, dnombre";
// visualizamos la sql para comprobar que se ha construido bien
System.out.println(sql);
// Creamos la instruccion
Statement sentencia= miconexion.createStatement();
// La ejecutamos recogiendo el nº de filas afectadas
int filas=sentencia.executeUpdate(sql);
// Visualizamos el nº de filas afectadas
System.out.println("Resultado de la operación: " + filas);
// Liberamos recursos
sentencia.close();
miconexion.close();
```

1.3. PreparedStatement

Hereda de java.sql.Statement. Se diferencia de java.sql.Statement en dos cosas:

- La sentencia SQL es compilada por el SGBD previamente.
- La sentencia puede contener parámetros (variables marcadas con ?).

Siempre lleva asociada una conexión que sirvió como origen para su creación.

Se crean con el siguiente método de la clase java.sql.Connection:

```
public PreparedStatement prepareStatement(String sql) throws SQLException;
```

Para ejecutar la sentencia se pueden utilizar los mismos métodos que para Staement pero sin cadena sql ya que la sql la habremos rellenado con el método prepareStatement():

```
public ResultSet executeQuery() throws SQLException;
public ResultSet executeUpdate() throws SQLException;
```

En el caso de que la sentencia contenga algún parámetro, el valor de dicho parámetro se tendrá que asignar antes de ejecutar la sentencia con el método:

```
public void setXXXX(int col, xxxx value) throws SQLException;
```

El método setXXXX maneja los mismos tipos que el método getXXXX visto en el punto anterior.

Las sentencias preparadas se cierran mediante el método:

```
public void close() throws java.sql.SQLException;
```

Lo mejor será verlo con un ejemplo:

```
Class.forName("com.mysql.jdbc.Driver");
// Establecemos la conexión con la base de datos
Connection miconexion = DriverManager.getConnection("jdbc:mysql://localhost/Ejemplo", "root", "root");
/ Recuperamos los argumentos del main()
System.out.println(args.length);
String dep=args[0];
String dnombre=args[1];
String loc=args[2];
// construimos la orden INSERT
String sql= "INSERT INTO departamentos VALUES (?,?,?)";
// Creamos la instruccion
PreparedStatement sentencia= miconexion.prepareStatement(sql);
// Antes de ejecutarla rellenamos los parámetros
// Tenemos que seleccionar el .set correspondiente al tipo de dato de la columna en la tabla
sentencia.setInt(1, Integer.parseInt(dep));
sentencia.setString(2, dnombre);
sentencia.setString(3, loc);
// La ejecutamos recogiendo el nº de filas afectadas
int filas=sentencia.executeUpdate();
/ Visualizamos el nº de filas afectadas
System.out.println("Nº de filas afectadas: " + filas);
// Liberamos recursos
sentencia.close();
miconexion.close();
```

Fíjate que tenemos en la sentencia SQL un ? por cada parámetro que se vaya a utilizar, dichos parámetros se identifican por su posición dentro de la sentencia empezando a contar en 1. Sólo se puede usar para representar valores, no identificadores, es decir que no se puede utilizar ? en lugar del nombre de la tabla en la FROM o en lugar de un nombre de columna.

Estas cadenas sql serían incorrectas:

"SELECT * FROM ?;" "SELECT INTO tabla (?,?,?) VALUES (1,'un valor', 10);"

Se crea la sentencia con el método prepareStatement() de la conexión.

Antes de ejecutar la sentencia nos tenemos que asegurar de haber rellenado los parámetros con los métodos .setXXX correspondientes.

Se ejecuta en este caso con .executeUpdate() porque la sql es un INSERT, si hubiera sido una SELECT hubiéramos utilizado .executeQuery() y el resultado devuelto lo hubiéramos asignado a un ResultSet.

La sentencia se puede ejecutar varias veces cambiando las asignaciones a los parámetros sin tener que volver a compilar la sentencia (con el método .prepareStatement()).

A la hora de asignar valores a los parámetros habrá que tener en cuenta el tipo de dato que se pasan para no generar errores. Por eso en el ejemplo parseamos a entero el valor de la variable dep para asegurarnos de pasar un entero.

Aquí tienes otro ejemplo con una SELECT:

```
Class.forName("com.mysql.jdbc.Driver");
// Establecemos la conexión con la base de datos
Connection miconexion = DriverManager.getConnection("jdbc:mysql://localhost/Ejemplo", "root", "root");
/ Recuperamos los argumentos del main()
System.out.println(args.length);
String dep=args[0];
String oficio=args[1];
// construimos la orden INSERT
String sql= "SELECT apellido, salario FROM empleados WHERE dept_no=? AND oficio=?";
// Creamos la instruccion
PreparedStatement sentencia= miconexion.prepareStatement(sql);
// Antes de ejecutarla rellenamos los parámetros
// Tenemos que seleccionar el .set correspondiente al tipo de dato de la columna en la tabla
sentencia.setInt(1, Integer.parseInt(dep));
sentencia.setString(2, oficio);
// La ejecutamos recogiendo el resultSet
ResultSet resultado =sentencia.executeQuery();
/ Visualizamos las filas devueltas
while (resultado.next()){
    System.out.println(resultado.getString("apellido")+ " --> " + resultado.getFloat("salario"));
}
// Liberamos recursos
sentencia.close();
miconexion.close();
```

También se puede utilizar sin parámetros, en este caso se utiliza si la instrucción se va a ejecutar varias veces en la misma conexión para mejorar el rendimiento.

1.4. CallableStatement

También hereda de java.sql.PreparedStatement, y sirve para ejecutar procedimientos almacenados en la Base de Datos.

Un procedimiento almacenado es un programa que reside y se ejecuta en el propio SGBD. Se puede definir con parámetros de entrada, con parámetros de salida, o ambos, o sin ninguno. También pueden devolver un valor, en este caso actúa como una función.

Como son definidos en el propio SGBD, los tipos de parámetros que admitan dependerán de lo que admita el SGBD.

Se crean con el siguiente método de la clase java.sql.Connection:

```
public Callablestatement prepareCall(String call) throws SQLException;
```

Las llamadas pueden tener tres formas:

- Con parámetros de entrada y de salida: {call nombre_procedimiento(?, ?, ...)}
- Con parámetros de entrada y es función: {? = call nombre_procedimiento(?, ?, ...)}
- Sin parámetros: {call nombre_procedimiento}

Para ejecutar la sentencia se llama al método correspondiente según el valor devuelto por el procedimiento:

```
public ResultSet executeQuery() throws SQLException;
public int executeUpdate() throws SQLException;
```

Antes de ejecutar la sentencia nos tenemos que asegurar de haber rellenado los parámetros con los métodos .setXXX correspondientes:

```
public void setXXXX(int col, xxxx value) throws SQLException;
```

Las sentencias se cierran mediante el método:

```
public void close() throws java.sql.SQLException;
```

Aquí tienes un ejemplo con un procedimiento que tiene dos parámetros de entrada:

```
// Cargar el driver
Class.forName("com.mysql.jdbc.Driver");
// Establecemos la conexión con la base de datos
Connection miconexion = DriverManager.getConnection("jdbc:mysql://localhost/Ejemplo", "root", "root");
// construimos la orden de llamada
String sql= "call subida_sal(?,?)";
// Creamos la instruccion
CallableStatement sentencia= miconexion.prepareCall(sql);
// Rellenamos los parámetros teniendo en cuenta los tipos de las columnas
sentencia.setInt(1,Integer.parseInt(args[0]));
sentencia.setDouble(2,Double.parseDouble(args[1]));
// La ejecutamos en este caso como es un procedimiento sin parámetros de salida no tenemos nada que rec
sentencia.executeUpdate();
// Visualizamos el nº de filas afectadas
System.out.println("Actualización realizada");
// Liberamos recursos
sentencia.close();
miconexion.close();
```

Fíjate en el método .prepareCall() que sirve para crear instanciar un objeto CallableStatement.

Otro ejemplo con varios procedimientos:

```
int dep = Integer.parseInt(args[0]);
// Cargar el driver
Class.forName("com.mysql.jdbc.Driver");
// Establecemos la conexión con la base de datos
Connection miconexion = DriverManager.getConnection("jdbc:mysql://lccalhost/Ejemplo", "root", "root");
// construimos la orden de llamada
String sql= "call p_nombredep(?,?)";
// Creamos la instruccion
CallableStatement sentencia= miconexion.prepareCall(sql);
// Rellenamos los parámetros teniendo en cuenta los tipos de las columnas
sentencia.setInt(1,dep); // en la sentencia el primer parámetroo es el departamento y el segundo el param
sentencia.registerOutParameter(2,Types.VARCHAR);
// La ejecutamos en este caso como es un procedimiento sin parámetrcs de salida no tenemos nada que recoç
sentencia.executeUpdate();
// Visualizamos el nº de filas afectadas
System.out.println("Según procedimiento, el departamento " + dep + " se denomina: " + sentencia.getStrinç
// Para ejecutar la función
sql= "{? = CALL f_nombredep(?)}"; // En este caso si no incluimos entre llaves, da error
// Creamos la instruccion
sentencia= miconexion.prepareCall(sql);
// Rellenamos los parámetros teniendo en cuenta los tipos de las columnas
sentencia.registerOutParameter(1,Types.VARCHAR); // Ahora el primer parámetro de sentencia es la variable
sentencia.setInt(2,dep);

// La ejecutamos en este caso como es un procedimiento sin parámetrcs de salida no tenemos nada que recoç
sentencia.executeUpdate();
System.out.println("Según función, el departamento " + dep + " se denomina: " + sentencia.getString(1));
```

El primer procedimiento que se ejecuta (*p_nombredep*) tiene un parámetro de entrada y uno de salida. Lo puedes ver por cómo se tratan las variables del objeto CallableStatement.

NOTA. Como aquí estamos utilizando la palabra *parámetro* para dos cosas distintas, utilizaremos la palabra *parámetro* para referirnos a los parámetros del procedimiento (los que están dentro de la definición del procedimiento) y la palabra *variable* para referirnos a los parámetros del Statement (los ??? que aparecen en la cadena sql).

A la primera variable se le asigna un entero con el valor del parámetro de entrada (sentencia.**setInt**(**1**,dep);) y a la segunda variable se le registra como un parámetro de salida (sentencia.**registerOutParameter**(**2**, Types.VARCHAR);).

Después de ejecutar la sentencia tenemos en la **segunda** variable el valor devuelto por el procedimiento a través de su parámetro de salida y lo podemos recuperar con sentencia.getString(**2**).

El segundo procedimiento *(f_nombredep*) es una función que devuelve una cadena y tiene un parámetro de entrada. La cadena sql que se ejecuta tiene dos variables, la primera recogerá el valor devuelto por la función y la segunda contendrá el valor del parámetro de entrada:

Sql="{? = f_nombredep(?)}"

En este caso el **primer** ? se refiere a la variable que va a recibir el valor devuelto por la función por eso se le define como un parámetro de salida (sentencia.registerOutParameter(**1**,Types.VARCHAR);).

La **segunda** variable (el segundo ?) contiene el valor que se le pasa a la función como parámetro de entrada por lo que se le asigna valor con (sentencia.setInt(**2**, dep);)).

Después de ejecutar la sentencia tenemos el valor devuelto en la primera variable por eso lo recuperamos con sentencia.getString(1);

Con esto terminamos con nuestro estudio de las interfaces Statement.

2. Gestión de errores

Habrás podido observar que todos los métodos que hemos manejado lanzan en caso de error una SQLException. Cuando se produce un error de tipo SQLException podemos acceder a cierta información usando los métodos:

- **.getMessage():** Devuelve una cadena que describe el error.
- **.getSQLState()**: cadena que contiene un estado definido por el estándar X/OPEN SQL.
- **.getErrorCode():** entero código del error según fabricante. Normalmente este será el código de error devuelto por el SGBD.

Se pueden utilizar para gestionar mejor los errores que puedan aparecer.