Material descargable

egg

Java Database Connectivity (JDBC)

En este capítulo, vas a explorar cómo interactuar con bases de datos en Java, utilizando la API JDBC. Aprenderás a crear objetos Statement para ejecutar consultas, a recuperar datos con ResultSet y a optimizar el rendimiento con PreparedStatement.

Recuperar datos de la base de datos. El objeto Statement

Una vez que tengas la conexión con el objeto **`Connection`**, la vas a usar para crear un objeto **`Statement`**. Este objeto recibe la consulta para ejecutarla y enviársela a la base de datos.

La información que recibas de la base de datos, va a ser capturada por el objeto **`ResultSet`** para después poder mostrarla.

```
try (Connection connection = DriverManager.getConnection(url, "user", "password");
     Statement stmt = connection.createStatement();
     ResultSet rs = stmt.executeQuery("SELECT a, b, c FROM Table1")) {

    while (rs.next()) {
        int x = rs.getInt("a");
        String s = rs.getString("b");
        double d = rs.getDouble("c");
        System.out.println("Fila = " + x + " " + s + " " + d);
    }
} catch (SQLException e) {
    e.printStackTrace();
}
```

Al usar esta sintaxis, Java garantiza que estos recursos se cerrarán automáticamente al final del bloque try, incluso si ocurre una excepción. Tambien podemos cerrar los recursos de manera manual.

Creación y ejecución de sentencias SQL (Objeto Statement)

Una vez que tengas la conexión a la base de datos, la podrás utilizar para crear sentencias. Estas sentencias están encapsuladas en la clase `Statement`, y puedes crear un objeto `Statement` de la siguiente manera:

```
Statement stmt = con.createStatement();
```

Los objetos `Statement` te permiten realizar consultas SQL en la base de datos. Se obtienen a partir de un objeto `Connection`. Estos objetos tienen distintos métodos para ejecutar consultas:

- **`executeQuery()`**: Envía a la base de datos una sentencia SQL para recuperar datos y devolver un único objeto `ResultSet`. Se utiliza para leer datos, típicamente con sentencias `SELECT`.
- **`executeUpdate()`**: Se usa para realizar actualizaciones que no devuelven un `ResultSet`. Es útil para insertar, modificar o borrar datos, como en las sentencias `INSERT`, `UPDATE` o `DELETE`.

Una vez que tengas el objeto `Statement`, puedes ejecutar sentencias utilizando el método `executeQuery()`, al que le pasas la sentencia SQL que quieres ejecutar:

```
stmt.executeQuery(sentenciaSQL);
```

Estas sentencias se utilizan para consultar la base de datos. Las sentencias SQL que escribas serán las mismas que usarías en un manejador de base de datos, pero deben ir entre comillas dobles, ya que el objeto `Statement` recibe una cadena de texto (String) para las sentencias.

Ejemplo de una sentencia SQL:

```
String sentenciaSQL = "SELECT nombre, apellido FROM alumnos";
```

Como puedes ver, las sentencias tendrán la misma sintaxis que has estado usando, pero hay una diferencia al trabajar con datos de tipo `String` y `Date`. Dado que estos datos generalmente van entre comillas dobles en Java y SQL, y tu sentencia SQL ya está entre comillas dobles, deberás poner los datos entre comillas simples para diferenciarlos. Aquí tienes un ejemplo:

String:

```
"SELECT nombre, apellido FROM Alumnos WHERE nombre = 'Agustin'";
```

Date:

```
"SELECT nombre FROM Alumnos WHERE fechaNacimiento = '01-11-1990'";
```

Obtención de datos (Objeto ResulSet)

Para obtener datos almacenados en la base de datos, utilizarás una consulta SQL (query). Puedes ejecutar esta consulta usando el objeto `Statement`, con el método `executeQuery()`, al que le pasas una cadena con la consulta SQL.

```
ResultSet result = stmt.executeQuery(sentenciaSQL);
```

La consulta SQL devolverá una tabla que tendrá una serie de campos y un conjunto de registros, cada uno de los cuales consistirá en una tupla de valores correspondientes a los campos de la tabla. El objeto `ResultSet` te permite acceder a los datos de estas filas mediante un conjunto de métodos `get` que te permitirán obtener los valores de las diferentes columnas de la fila actual.

El método `ResultSet.next()` se usa para moverse a la siguiente fila del `ResultSet`, convirtiéndola en la fila actual. El formato general de un `ResultSet` es similar a una tabla, con cabeceras de columna y los valores correspondientes devueltos por la consulta. Por ejemplo, si la consulta es `SELECT a, b, c FROM Table1`, el resultado tendrá una forma como la siguiente:

a	b	c
12345	Argentina	10,5
31245	Brasil	22,7
47899	Perú	56,7

A continuación te dejo un fragmento de código como ejemplo. Este código ejecuta una consulta SQL que devuelve una colección de filas, con la columna 1 como un `int`, la columna 2 como un `String` y la columna 3 como un `double`:

```
Statement stmt = connection.createStatement();
ResultSet r = stmt.executeQuery("SELECT a, b, c FROM Table1");
while (r.next()) {
    int i = r.getInt("a");
    String s = r.getString("b");
    double d = r.getDouble("c");
    // Imprime los valores de la fila actual
    System.out.println("Fila = " + i + " " + s + " " + d);
}
```

Filas ResultSet

Un `ResultSet` mantiene un cursor que apunta a la fila actual de datos. El cursor se mueve una fila hacia abajo cada vez que se llama al método `next()`. Inicialmente, el cursor está antes de la primera fila, por lo que debes llamar al método `next()` para situarlo en la primera fila y convertirla en la fila actual. Las filas del `ResultSet` se recuperan de manera secuencial, de arriba hacia abajo.

Columnas ResultSet

Los métodos `get` te proporcionan la manera de recuperar los valores de las columnas de la fila actual. Los valores de las columnas pueden recuperarse en cualquier orden dentro de cada fila, pero para asegurar la máxima portabilidad, es recomendable extraer las columnas de izquierda a derecha y leer los valores de las columnas una única vez.

Puedes usar, tanto el nombre de la columna como su número de columna, para referirte a ella. Por ejemplo, si la segunda columna de un objeto `ResultSet` se llama "nombre" y almacena valores de tipo `String`, cualquiera de los dos ejemplos siguientes devolverá el valor almacenado en esa columna:

```
String s = rs.getString("nombre");
String s = rs.getString(2);
```

Ten en cuenta que las columnas se numeran de izquierda a derecha, comenzando desde la columna 1. Además, los nombres utilizados como input en los métodos `get` son insensibles a las mayúsculas.

Algunos de los métodos que puedes usar para obtener datos son:

Método	Explicación
getInt()	Sirve para obtener un número entero de la base de datos
getLong()	Sirve para obtener un número long de la base de datos
getDouble()	Sirve para obtener un número real de la base de datos
getBoolean()	Sirve para obtener un booleano de la base de datos
getString()	Sirve para obtener una cadena de la base de datos
getDate()	Sirve para obtener una fecha de la base de datos

De esta forma, podrás acceder a los datos de tu base de datos de manera eficiente y ordenada.

Optimización de sentencias

Cuando necesitas invocar una determinada sentencia SQL repetidas veces, puede ser conveniente prepararla de antemano para que se ejecute de forma más eficiente. Para esto, utilizarás la interfaz `PreparedStatement`, que puedes obtener de la conexión a la base de datos de la siguiente manera:

```
PreparedStatement ps = con.prepareStatement("SELECT * FROM nombreTabla WHERE campo2 > 1200 AND campo2 < 1300");
```

A diferencia del objeto `Statement`, que vimos anteriormente, a `PreparedStatement` le proporcionas la sentencia SQL en el momento de su

creación. De esta forma, la sentencia estará optimizada y lista para ejecutarse de manera más eficiente cada vez que la necesites.

Sin embargo, lo más común es que no necesites ejecutar exactamente la misma sentencia una y otra vez, sino que necesitarás hacer variaciones de ella. Por lo tanto, el objeto `PreparedStatement` te permite parametrizar la sentencia. Para ello, debes colocar el carácter `?` en lugar de los valores específicos, indicando la posición de los parámetros dentro de la consulta. Así:

```
PreparedStatement ps = con.prepareStatement("UPDATE nombreTabla
SET campo1 = 'valor' WHERE campo2 > ? AND campo2 < ?");
```

En este ejemplo, tienes dos parámetros que representarán un rango de valores en el cual deseas realizar la actualización. Cuando ejecutes esta sentencia, el campo `campo1` de la tabla `nombreTabla` se establecerá a "valor" para los registros cuyo `campo2` esté dentro del rango especificado.

Para asignar valores a esos parámetros, utilizarás el método `setXXX()`, donde `XXX` dependerá del tipo de dato que estés asignando. Debes indicar el número del parámetro (que empieza desde 1) y el valor que le quieres dar. Por ejemplo, para asignar valores enteros a los parámetros, lo harías de la siguiente manera:

```
ps.setInt(1, 1200);

ps.setInt(2, 1300);
```

Una vez asignados los valores a los parámetros, puedes ejecutar la sentencia llamando al método `executeUpdate()` del objeto `PreparedStatement`:

```
int n = ps.executeUpdate();
```

Al igual que con los objetos `Statement`, también puedes usar otros métodos para ejecutar la sentencia, como `executeQuery()` o `execute()`, dependiendo del tipo de sentencia que estés ejecutando.

Para quienes deseen explorar más sobre los componentes del API de JDBC y su funcionamiento, recomendamos consultar la **documentación oficial** proporcionada por Oracle. Allí encontrarán información detallada, ejemplos y guías actualizadas para implementar JDBC en sus proyectos.