Consultas a una BD con JDBC

Índice

1. Restricciones, movimientos y actualizaciones en el ResultSet

Cuando realizamos llamadas a BD de gran tamaño el resultado de la consulta puede ser demasiado grande y no deseable en términos de eficiencia y memoria. JDBC permite restringir el número de filas que se devolverán en el **ResultSet**. La clase **Statement** incorpora dos métodos, **getMaxRows** y **setMaxRows**, que permiten obtener e imponer dicha restricción. Por defecto, el límite es cero, indicando que no se impone la restricción. Si, por ejemplo, antes de ejecutar la consulta imponemos un límite de 30 usando el método **setMaxRows(30)**, el resultado devuelto sólo contendrá las 30 primeras filas que cumplan con los criterios de la consulta.

Hasta ahora, el manejo de los datos devueltos en una consulta se realizaba con el método **next** de **ResultSet**. Podemos manejar otros métodos para realizar un movimiento no lineal por el **ResultSet**. Es lo que se conoce como **ResultSet** arrastable. Para que esto sea posible debemos utilizar el siguiente método en la creación del **Statement**:

```
Statement createStatement (int resultSetType, int resultSetConcurrency)
```

Los posibles valores que puede tener **resultSetType** son: **ResultSet.TYPE_FORWARD_ONLY**, **ResultSet.TYPE_SCROLL_INSENSITIVE**, **ResultSet.TYPE_SCROLL_SENSITIVE**. El primer valor es el funcionamiento por defecto: el **ResultSet** sólo se mueve hacia adelante. Los dos siguientes permiten que el resultado sea arrastable. Una característica importante en los resultados arrastables es que los cambios que se produzcan en la BD se reflejan en el resultado, aunque dichos cambios se hayan producido después de la consulta. *Esto dependerá de si el driver y/o la BD soporta este tipo de comportamiento*. En el caso de **INSENSITIVE**, el resultado no es sensible a dichos cambios y en el caso de **SENSITIVE**, sí. Los métodos que podemos utilizar para movernos por el **ResultSet** son:

next	Pasa a la siguiente fila
previous	Ídem fila anterior
last	Ídem última fila
first	Ídem primera fila
absolute(int fila)	Pasa a la fila número fila
relative(int fila)	Pasa a la fila número fila desde la actual
getRow	Devuelve la número de fila actual
isLast	Devuelve si la fila actual es la última

isFirst	Ídem la primera

El otro parámetro, **resultSetConcurrency**, puede ser uno de estos dos valores: **ResultSet.CONCUR_READ_ONLY** y **ResultSet.CONCUR_UPDATABLE**. El primero es el utilizado por defecto y no permite actualizar el resultado. El segundo permite que los cambios realizados en el **ResultSet** se actualicen en la base de datos. Si queremos modificar los datos obtenidos en una consulta y queremos reflejar esos cambios en la BD debemos crear una sentencia con **TYPE_FORWARD_SENSITIVE** y **CONCUR_UPDATABLE**.

Para actualizar un campo disponemos de métodos **updateXXXX**, de la misma forma que teníamos métodos **getXXXX**. Estos métodos reciben dos parámetros: el primero indica el nombre del campo (o número de orden dentro del **ResultSet**); el segundo indica el nuevo valor que tomará el campo del registro actual. Para que los cambios tengan efecto en la BD debemos llamar al método **updateRow**. El siguiente código es un ejemplo de modificación de datos:

```
rs.updateString("nombre","manolito");
rs.updateRow();
```

Si queremos desechar los cambios producidos en la fila actual (antes de llamar a **updateRow**) podemos llamar a **cancelRowUpdates**. Para borrar la fila actual tenemos el método **deleteRow**. La llamada a este método deja una fila vacía en el **ResultSet**. Si intentamos acceder a los datos de esa fila nos dará una excepción. Podemos llamar al método **rowDeleted** el cual devuelve cierto si la fila actual ha sido eliminada (método no implementado en MySQL).

Debemos tener en cuenta varias restricciones a la hora de actualizar un **ResultSet**: la sentencia SELECT que ha generado el **ResultSet** debe:

- Referenciar sólo una tabla.
- No contener una claúsula *join* o *group by*.
- Seleccionar la clave primaria de la tabla.

Existe un registro especial al que no se puede acceder como hemos visto anteriormente, que es el registro de inserción. Este registro se utiliza para insertar nuevos registros en la tabla. Para situarnos en él deberemos llamar al método **moveToInsertRow**. Una vez situados en él deberemos asignar los datos con los métodos **updateXXXX** anteriormente descritos y una vez hecho esto llamar a **insertRow** para que el registro se inserte en la BD. Podemos volver al registro donde nos encontrábamos antes de movernos al registro de inserción llamando a **moveToCurrentRow**.

2. Sentencias de actualización

La clase **statement** dispone de un método llamado **executeUpdate** el cual recibe como parámetro la cadena de caracteres que contiene la sentencia SQL a ejecutar. Este método únicamente permite realizar sentencias de actualización de la BD: creación de tablas (CREATE), inserción (INSERT), actualización (UPDATE) y borrado de datos (DELETE). El método a utilizar es el siguiente:

```
stmt.executeUpdate(sentencia);
```

Vamos a ver a continuación un ejemplo de estas operaciones. Crearemos una tabla ALUMNOS en nuestra base de datos y añadiremos datos a la misma. La sentencia para la creación de la tabla será la siguiente:

```
String st_crea = "CREATE TABLE ALUMNOS (
    exp INTEGER,
    nombre VARCHAR(32),
    sexo CHAR(1),
    PRIMARY KEY (exp)
)";
stmt.executeUpdate(st_crea);
```

Una vez creada la tabla podremos insertar datos en ella como se muestra a continuación:

```
String st_inserta = "INSERT INTO ALUMNOS(exp, nombre)
    VALUES(1285, 'Manu', 'M')";
stmt.executeUpdate(st_inserta);
```

Cuando tengamos datos dentro de la tabla, podremos modificarlos utilizando para ello una sentencia UPDATE:

```
String st_actualiza = "UPDATE FROM ALUMNOS
    SET sexo = 'H' WHERE exp = 1285";
stmt.executeUpdate(st_actualiza);
```

Si queremos eliminar un registro de la tabla utilizaremos una sentencia DELETE como se muestra a continuación:

```
String st_borra = "DELETE FROM ALUMNOS
    WHERE exp = 1285";
stmt.executeUpdate(st_borra);
```

El método **executeUpdate** nos devuelve un entero que nos dice el número de registros a los que ha afectado la operación, en caso de sentencias INSERT, UPDATE y DELETE. La creación de tablas nos devuelve siempre 0.

3. Otras llamadas a la BD

En la interfaz **Statement** podemos observar un tercer método que podemos utilizar para la

ejecución de sentencias SQL. Hasta ahora hemos visto como para la ejecución de sentencias que devuelven datos (consultas) debemos usar **executeQuery**, mientras que para las sentencias INSERT, DELETE, UPDATE e instrucciones DDL utilizamos **executeUpdate**. Sin embargo, puede haber ocasiones en las que no conozcamos de antemano el tipo de la sentencia que vamos a utilizar (por ejemplo si la sentencia la introduce el usuario). En este caso podemos usar el método **execute**.

```
boolean hay_result = stmt.execute(sentencia);
```

Podemos ver que el método devuelve un valor *booleano*. Este valor será *true* si la sentencia ha devuelto resultados (uno o varios objetos **ResultSet**), y *false* en el caso de que sólo haya devuelto el número de registros afectados. Tras haber ejecutado la sentencia con el método anterior, para obtener estos datos devueltos proporciona una serie de métodos:

```
int n = stmt.getUpdateCount();
```

El método **getUpdateCount** nos devuelve el número de registros a los que afecta la actualización, inserción o borrado, al igual que el resultado que devolvía **executeUpdate**.

```
ResultSet rs = stmt.getResultSet();
```

El método **getResultSet** nos devolverá el objeto **ResultSet** que haya devuelto en el caso de ser una consulta, al igual que hacía **executeQuery**. Sin embargo, de esta forma nos permitirá además tener múltiples objetos **ResultSet** como resultado de una llamada. Eso puede ser necesario, por ejemplo, en el caso de una llamada a un procedimiento, que nos puede devolver varios resultados como veremos más adelante. Para movernos al siguiente **ResultSet** utilizaremos el siguiente método:

```
boolean hay_mas_results = stmt.getMoreResults();
```

La llamada a este método nos moverá al siguiente **ResultSet** devuelto, devolviéndonos *true* en el caso de que exista, y *false* en el caso de que no haya más resultados. Si existe, una vez nos hayamos movido podremos consultar el nuevo **ResultSet** llamando nuevamente al método **getResultSet**.

Otra llamada disponible es el método **executeBatch**. Este método nos permite enviar varias sentencias SQL a la vez. No puede contener sentencias SELECT. Devuelve un array de enteros que indicará el número de registros afectados por las sentencias SQL. Para añadir sentencias haremos uso del método **addBatch**. Un ejemplo de ejecución es el siguiente:

```
stmt.addBatch("INSERT INTO ALUMNOS(exp, nombre)
    VALUES(1285, 'Manu', 'M')");
stmt.addBatch("INSERT INTO ALUMNOS(exp, nombre)
    VALUES(1299, 'Miguel', 'M')");

int[] res = stmt.executeBatch();
```