PROGRAMACIÓN UD-14

ACCESO A DATOS

ÍNDICE

[INTRODUCCIÓN 3](#_Toc134727960)

[CONEXIÓN 3](#_Toc134727961)

[EJECUCIÓN DE SENTENCIAS 3](#_Toc134727962)

[RESULTSET 4](#_Toc134727963)

[SQL INJECTION 6](#_Toc134727964)

[PREPAREDSTATEMENT 6](#_Toc134727965)

[OPERACIONES CRUD 7](#_Toc134727966)

[DAO 7](#_Toc134727967)

# INTRODUCCIÓN

Hasta ahora, hemos utilizado ficheros para guardar información de manera permanente, pero si el volumen de acceso a la cantidad de datos es elevado, es interesante almacenar los datos en bases de datos.

Para la conexión al SGBD usaremos la API Java. SQL que contiene las clases:

* DriverManager: indicamos al SGBD con el que nos vamos a conectar.
* Connection: crea una conexión con el SGBD.
* Statement y PreparedStatement: indica la sentencia a ejecutar en el SGBD.
* ResulSet: es el cursor resultante de la sentencia.

# CONEXIÓN

Cada fabricante desarrolla un driver JDBC que conoce a las peculiaridades del sistema gestor de bases de datos asociado, haciendo de puente o traductor entre el resto de las clases de la API de JDBC y el servidor de la base de datos.

Los drivers los deberemos descargar del fabricante específico de sistema gestor de bases de datos.

Antes de empezar a usar la base de datos, hay que establecer una conexión (usamos la clase Connection) con la misma. A través de esa conexión nos iremos conectando con la base de datos. Para crear un objeto connection tenemos que indicar la url, donde indicaremos la base de datos a la que deseamos conectarnos, el usuario y la password.

El formato de la URL será “jdbc:mysql//Servidor:puerto/BD”.



Las excepciones del tipo SQLException (si pasa demasiado tiempo sin poder conectarse, puede enviar una subexcepción SQLTimeoutException).

# EJECUCIÓN DE SENTENCIAS

Para ejecutar una sentencia SQL preestablecida, usaremos un objeto de la clase Statement, que contendrá un objeto SQL que queramos ejecutar.

Lo primero se crea el objeto Statement en base al objeto Connection creado con anterioridad. Statement sql = conexión.createStatement();

Posteriormente crearemos un cursor que almacenará el resultado de la consulta ResultSet rs = sql.executeQuery(“SELECT \* FROM PRUEBA”); puede lanzar excepción SQLException o SQLTimeoutException.

Si lo que queremos ejecutar es un INSERT, un UPDATE o un DELETE, ejecutaremos el método executeUpdate, que nos devuelve el número de filas que han sido afectadas (sql.executeUpdate(“UPDATE PRUEBA SET NOMBRE = ‘JUAN’ WHERE NOMBRE = ‘PEPE’”).

# RESULTSET

Una vez ejecutado el método executeQuery(), podremos coger del ResultSet los diferentes valores, bien indicando el nombre del con un String o indicando con un número el atributo según el orden de petición (normalmente menos recomendado). Existe un get para cada tipo de datos, aunque es más recomendable recoger los valores con un String y luego hacer un parse al tipo de datos concreto.

rs.getString("nombre"); o rs.getString(1);

rs.getInt("edad"); o rs.getDouble("3");

Al ejecutar la consulta, nos situamos con el ResultSet en el primer registro de salida de la consulta. Con el método next() nos podemos desplazar al siguiente registro, si puede devuelve true y si no hay más registro devolverá false.

Se pueden usar ResultSet que permitan la modificación del registro actual y desplazamiento a voluntad del cursor. Para poder conseguirlo, al definir el Statement hay que indicar el tipo de ResultSet y la concurrencia.

Tipos:

* ResultSet.TYPE\_FORWARD\_ONLY: movimiento hacia adelante únicamente de un registro (por defecto)
* ResultSet.TYPE\_SCROLL\_INSENSITIVE: movimiento hacia
* delante y atrás. No se ven los datos en tiempo real de la BD, sino los que se tenían cuando se ejecutó la consulta
* ResultSet.TYPE\_SCROLL\_SENSITIVE: igual que el anterior, pero los datos se actualizan si hay cambios en la BD

Concurrencia:

* ResultSet.CONCUR\_READ\_ONLY: los datos son de solo lectura (por defecto)
* ResultSet.CONCUR\_UPDATABLE: es posible modificar los datos del ResultSet

Para modificar los datos de los atributos utilizaremos el método rs.update Tipo("columna", valor) o rs.update Tipo(no, valor). Tras modificar los atributos que nos interese hay que ejecutar rs.updateRow(); ES NECESARIO QUE EXISTA UNA PRIMARY KEY EN LA TABLA PARA QUE FUNCIONE.

Es posible eliminar el registro en el que estemos ejecutando el método result.deleteRow();

Para insertar un nuevo registro, nos moveremos a un registro nuevo, modificaremos sus datos, lo guardaremos y posteriormente volveremos a la posición en la que estábamos. Si no modificamos algún atributo quedará como null, pero por lo menos hay que cambiar la primary key

Texto

Descripción generada automáticamente

Los siguientes métodos se pueden utilizar con ResultSet que permitan el desplazamiento en ambos sentidos. Todos los métodos devuelven un booleano, que indica si ha sido posible realizar el desplazamiento.

* next(): avanza al siguiente registro
* previous(); retrocede al anterior registro
* first(): se desplaza al primer registro
* last(): se desplaza al último registro
* absoute(int): se desplaza al registro específicamente indicado (1 primero, 2 segundo... -1 último, -2 penúltimo...)
* relative(int): se mueve al registro que está a tantos registros como indiquemos (1 siguiente, -1 anterior, O no se desplaza...)
* beforeFirst(): se coloca antes del primer registro
* afterLast(): se coloca tras el último registro
* move ToInsertRow(): nos colocamos en un registro nuevo
* moveToCurrentRow(): volvemos a la posición que estábamos tras hacer una nueva inserción.

Existen otros métodos que devuelven un booleano en función de la posición donde se encuentra el cursor.

* isFirst( ): devuelve true si se está apuntando al primer registro
* isLast(): devuelve true si se está apuntando al último registro
* isBeforeFirst(): devuelve true si se está apuntando al registro anterior al primero
* isAfterLast(): devuelve true si se está apuntando al registro posterior al último

# SQL INJECTION

SQL Injection es una técnica de hacking muy conocida, que consiste en insertar código SQL en los datos que nos pida la aplicación y así conseguir resultados diferentes a lo esperado.

Por ejemplo si sabemos que tenemos una aplicación que elimina un usuario y por tanto pido cual es el usuario que deseamos eliminar y la aplicación ejecuta el siguiente string "DELÊTE FROM alumnos WHERE apellido=""+strNombre+"'", si strNombre contine el nombre de una persona, borrará a esa persona, pero si contine xxx' OR 'a'='a, el resultado final a ejecutar será "DELETE FROM alumnos WHERE apellido='xxx' OR 'a'='a"" con lo que eliminaremos toda los registros de la tabla.

En el caso de executeUpdate() solo se permite enviar una sentencia a la base de datos, pero en otros lenguajes se pueden insertar varias, y podríamos haber puesto en el campo de nombre varias sentencias...

# PREPAREDSTATEMENT

Cuando vamos a invocar una determinada sentencia repetidas veces, puede ser conveniente dejar esa sentencia preparada para que pueda ser ejecutada de forma más eficiente. Para hacer esto utilizaremos la interfaz PreparedStatement, que podrá obtenerse a partir de la conexión a la BD.

Vemos que a este objeto, a diferencia del objeto Statement visto anteriormente, le proporcionamos la sentencia SQL en el momento de su creación, por lo que estará preparado y optimizado para la ejecución de dicha sentencia posteriormente.

Sin embargo, lo más común es que necesitemos hacer variaciones sobre la sentencia, ya que normalmente no será necesario ejecutar repetidas veces la misma sentencia exactamente, sino variaciones de ella. Por ello, este objeto nos permite parametrizar la sentencia. Estableceremos las posiciones de los parámetros con el carácter '?' dentro de la cadena de la sentencia.

En este caso tenemos dos parámetros, que será el número de expediente mínimo y el máximo del rango que queremos actualizar. Cuando ejecutemos esta sentencia, la nota de los alumnos desde expediente inferior hasta expediente superior se establecerá a NO PRESENTADO ('NP').

Para dar valor a estos parámetros utilizaremos método setXXX() donde XXX será el tipo de los datos que asignamos al parámetro, indicando el número del parámetro (que empieza desde 1) y el valor que le queremos dar.

Igual que en el caso de los objetos Statement, podremos utilizar cualquier otro de los métodos para la ejecución de sentencias, executeQuery() o execute(), según el tipo de sentencia que vayamos a ejecutar.

# OPERACIONES CRUD

CRUD es un acónimo de las operaciones básicas de una BD (crear, leer, actualizar y borrar).

Para poder almacenar objetos en la BD, vamos a utilizar una técnica llamada mapeo objeto- relacional, que consiste mapear o vincular cada atributo de una clase con un campo de una tabla de la BD. Cada registro edad: 27 de una tabla será un objeto y la tabla será la clase.

Para cada clase de nuestra aplicación, para la cual queramos almacenar objetos en la BD, deberemos añadir los siguientes métodos en la propia clase:

* create(): inserta los datos del objeto en la BD mediante un insert
* read(): asignará al objeto que hace la llamada (el objeto tiene que tener en sus atributos la pk), los datos del registro de la BD asociados a esa pk con un select
* update(): modifica los datos del registro de la BD, que tenga como pk, la que tenga el objeto de la llamada con un update
* delete(): elimina el registro de la BD, que tenga como pk, la que tenga el objeto de la llamada, con un delete

# DAO

Para poder implementar CRUD, hemos tenido que implementar una serie de métodos que no se corresponden al mundo de la clase.

Existe otro enfoque llamado DAO (objeto de acceso a dato), que creará un objeto específico para las operaciones sobre la base de datos.