CONEXIÓN A BASES DE DATOS RELACIONALES CON JAVA: JDBC (MYSQL)

Programación

18 DE FEBRERO DE 2025

ERIK AMO TOQUERO

Contenido

Introducción

Las aplicaciones, tras coger una serie de datos, los procesan y generan nuevos datos. Estos datos necesitan ser almacenador de forma permanente para su posterior uso. Se pueden hacer de dos formas:

- Ficheros (en desuso)
- Bases de Datos

La API de JDBC del paquete de java.sql está compuesto de clases que trabajan de forma conjunta. Proporcionan métodos para acceder y manipular datos almacenados en una base de datos relacional. JDBC permite a los desarrolladores escribir aplicaciones en Java que pueden interactuar con bases de datos a través del estándar SQL.

Algunas de ellas son:

- DriverManager: Manipula drivers para acceder a distintos gestores de bases de datos
- Connection: Permite la conexión a bases de datos
- Statement : Esta interfaz se utiliza para ejecutar declaraciones SQL simples sin parámetros. Es útil para realizar operaciones como consultas, actualizaciones, inserciones o eliminaciones en la base de datos.
- PreparedStatement : Similar a Statement, pero precompila la sentencia SQL antes de ejecutarla. Esto puede mejorar el rendimiento cuando se ejecuta la misma sentencia varias veces con diferentes valores de parámetros.
- ResultSet : Esta interfaz representa un conjunto de resultados de una consulta SQL. Permite navegar a través de los registros devueltos por una consulta y recuperar los valores de las columnas.

Conexión

Antes de interactuar con una base de datos, es fundamental establecer una conexión entre nuestra aplicación y el Sistema de Gestión de Bases de Datos (SGBD). La conexión debe mantenerse abierta mientras se necesite y cerrarse adecuadamente cuando ya no se requiera.

Método para Establecer la Conexión.

Para crear una conexión, utilizamos el método estático DriverManager.getConnection(). Este método toma la URL de la base de datos, el nombre de usuario y la contraseña como parámetros y devuelve un objeto Connection.

```
private static Connection conectar() {
    Connection con = null; // SE INICIALIZA EN NULL
    try {
        String url = "jdbc:mysql://localhost:3306/_nombre de la base de datos_";
        con = DriverManager.getConnection(url, "_nombre del usuario_", "_contraseña del usuario_");
        System.out.println("Conectado correctamente"); // MENSAJE PARA CONFIRMAR QUE SE HA CONECTADO CORRECTAMENTE
    } catch (SQLException e) {
        System.out.println("Error al conectarme a la base de datos " + e);
    }
    return con;
}
```


Manejo de excepciones.

Es importante capturar y manejar adecuadamente las excepciones que pueden ocurrir durante la conexión. En el ejemplo proporcionado, se utiliza un bloque try-catch para manejar cualquier posible error.

También se puede optar por declarar que el método conectar() lanza una excepción SQLException, lo que permite manejar la excepción en un nivel superior, es decir, que el método no se encarga de manejar la excepción internamente, sino que la pasa a la parte del código que llama a ese método.

```
private static Connection conectar() throws SQLException {
    Connection con = null;
    try {
        String url = "jdbc:mysql://localhost:3306/_nombre de la base de datos_";
        con = DriverManager.getConnection(url, "_nombre del usuario_", "_contraseña del usuario_");
        System.out.println("Conectado correctamente"); // MENSAJE PARA CONFIRMAR QUE SE HA CONECTADO CORRECTAMENTE
    } catch (SQLException e) {
        System.out.println("Error al conectarme a la base de datos " + e);
    }
    return con;
}
```

Cerrar conexión.

Cuando ya no se requiere la conexión, se cierra.

```
Connection con = conectar(); // LLAMO AL MÉTODO PARA CONECTARME A LA DB
con.close(); // CIERRO LA CONEXIÓN
```

Ejecución de sentencias .

Ejecución de consultas SELECT.

Primero se crea un Statement con createStatement, y después, el método executeQuery() de Statement ejecuta una consulta y devuelve el resultado de esta mediante un objeto de tipo ResulSet.

```
Statement statement = connection.createStatement(); // STATEMENT
String sqlQuery = "SELECT * FROM _mytable_"; // GUARDAMOS EN UN STRING LA CONSULTA QUE QUEREMOS HACER
ResultSet resultSet = statement.executeQuery(sqlQuery) // SE EJECUTA Y DEVUELVE LA CONSULTA
```

Ejecución de sentencias : INSERT, UPDATE, DELETE.

Estas ejecuciones no devuelven un resultado, sino que realizan una operación.

Primero se crea un Statement con createStatement, y después, el método executeUpdate() de Statement devuelve el número de filas que se han modificado.

```
Statement statement = connection.createStatement(); // STATEMENT
String sqlUpdate = "UPDATE _mytable_ SET _column1_ = 'new value' WHERE _id_ = 1";
int rowsUpdated = statement.executeUpdate(sqlUpdate) // NOS DEVUELVE EL NUMERO DE FILAS QUE HAN SIDO MODIFICADAS
```


Clases DAO

Los Objetos de Acceso a Datos (Data Access Object (DAO)) son clases con métodos estáticos hechas para realizar las operaciones CRUD pero con las bases de datos. Hay 5/6 métodos:

Método Connection

Este es susceptible a no aparecer en TODAS las clases, siempre que aparezca como mínimo en una. Sirve para crear la conexión a la base de datos que se va a gestionar y reutilizar el comando:

```
public static Connection connect(){
    Connection con;
    try {
        String nombreUsuario, contraseña, url;
        con = DriverManager.getConnection(url, nombreUsuario,
contraseña);
    }catch (SQLException e){
        con=null;
    }
    return con;
}
```

Es importante el try-catch, ya que con un throws puede dar problemas al devolver la conexión.

Método int Add

Sirve para utilizar el script INSERT INTO. La actualización devuelve SIEMPRE valores numéricos

```
public static int addIntoTable(Object o){
    int numFilas;
    String sql = "INSERT INTO TABLE VALUES(?, ?, ?)";
    try {
        Connection con = connect();

        PreparedStatement sentencia = con.prepareStatement(sql);
        sentencia.setInt(1, o.valor1);
        sentencia.setString(2, o.valor2);
        sentencia.setDate(3, o.valor3);

        numFilas = sentencia.executeUpdate();

        con.close();
    }catch (SQLException e){
        numFilas=0;
    }

    return numFilas;
}
```


Método Object find

Se utiliza el método SELECT para poder buscar un objeto específico

```
public static Object find(int id){
    Object o;
    String sql = "SELECT * FROM TABLE WHERE ID = ?"
    try {
        Connection con = connect();
        PreparedStatement sentencia = con.prepareStatement(sql);
        sentencia.setInt(1, id);

        ResultSet resultado = sentencia.executeQuery();

        if (resultado.next()){//Si encuentra resultado
            Tipo valor1 = resultado.getTipo(1);
            Tipo valor2 = resultado.getTipo(2);
            Tipo valor3 = resultado.getTipo("valor3");
            o = new Object(valor1, valor2, valor3);
        } else {
            o = null;
        }
        con.close();
    }catch (SQLException e){
        o=null;
    }

    return o;
}
```

Método List<Object> list

```
public static ArrayList<Object> list(int id){
    ArrayList<Object> listaObjeto = new ArrayList<>();
    Object o;
    String sql = "SELECT * FROM TABLE";
    try {
        Connection con = connect();
        PreparedStatement sentencia = con.prepareStatement(sql);

        ResultSet resultado = sentencia.executeQuery();

        while (resultado.next()){//Mientras encuentra resultado
            Tipo valor1 = resultado.getTipo(1);
            Tipo valor2 = resultado.getTipo(2);
            Tipo valor3 = resultado.getTipo("valor3");
            o = new Object(valor1, valor2, valor3);
            listaObjeto.add(o);
        }
        con.close();
    }catch (SQLException e){
        listaObjeto=null;
    }

    return listaObjeto;
}
```


Método int Remove

Se utiliza para borrar objetos de la tabla por su identificador.

```
public static int deleteFromTable(int id){
    int numFilas;
    String sql = "DELETE FROM TABLE WHERE ID = ?";
    try {
     Connection con = connect();

     PreparedStatement sentencia = con.prepareStatement(sql);
     sentencia.setInt(1, id);

     numFilas=sentencia.executeUpdate();

     con.close();
    }catch (SQLException e){
        numFilas=0;
    }
    return numFilas;
}
```

Método int Update

Se pasa un objeto entero y se actualizan sus objetos en función de algún identificador:

```
public static int updateItemInTable(Object o){
    String sql = "UPDATE TABLE SET VALOR1=?, VALOR2=?, VALOR3=? WHERE
ID=?";
    int numFilas;
    try {
        Connection con = connect();

        PreparedStatement sentencia = con.prepareStatement(sql);
        sentencia.setString(1, o.valor1);
        sentencia.setString(2, o.valor2);
        sentencia.setString(3, o.valor3);
        sentencia.setInt(4, o.id);

        numFilas = sentencia.executeUpdate();

        con.close();
    }catch (SQLException e){
        numFilas=0;
    }
    return numFilas;
}
```