



Tema 14: Diseño de programas para la gestión de bases de datos relacionales

¿Qué aprenderás?

- Crear aplicaciones que almacenen y trabajen con datos de una base de datos.
- Realizar operaciones de inserción, modificación y borrado sobre una base de datos.
- Recuperar los datos de una base de datos y trabajar con ellos desde Java.

¿Sabías que...?

- JDBC existe desde 1996 y fue incluido desde la versión 1.1 de JDK, en 1997.
- MySQL tiene doble licencia: GPL y Licencia comercial por Oracle Corporation. Con la compra de MySQL por parte de Oracle, surgió la base de datos MariaDB, derivación de MySQL que cuenta con la mayoría de sus características.



14. Diseño de programas para la gestión de bases de datos relacionales

Para que nuestros programas guarden los datos con los que trabajan hemos visto que podemos usar ficheros. Este sistema de almacenaje es fácil de usar y poco costoso, pero como inconvenientes cabe destacar la posible inconsistencia y redundancia de los datos. Es por esto, entre otros motivos más, que hoy en día la mayoría de aplicaciones que implementan la persistencia de datos utilicen bases de datos.

En este capítulo veremos las herramientas que nos proporciona Java para que nuestros programas accedan a la información contenida en una base de datos. Vamos a trabajar con MySQL, pero se puede trabajar con otras bases de datos. Cambiará la librería a usar, pero el código que veremos será el mismo.

14.1. La API JDBC

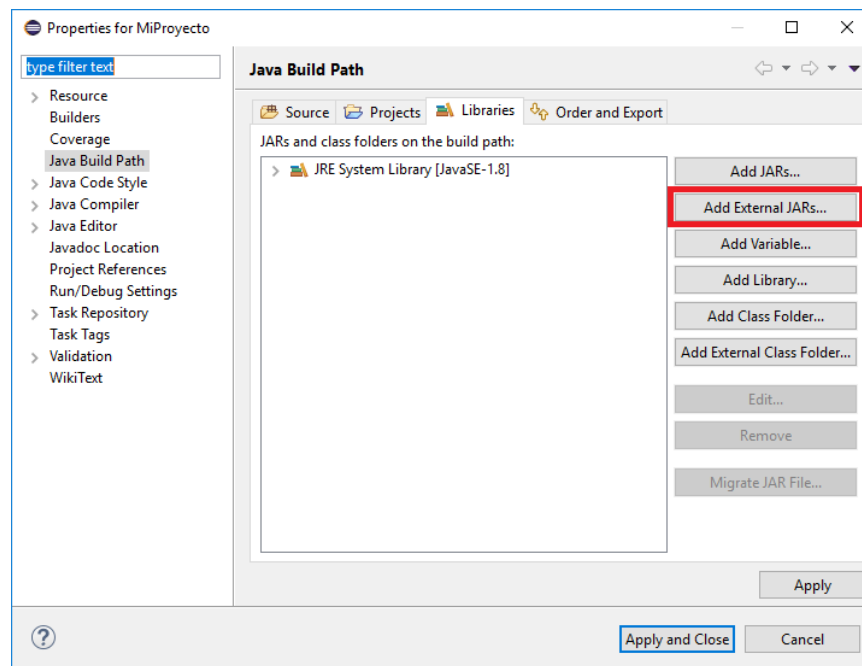
Para que nuestros programas puedan realizar operaciones con los datos almacenados en una base de datos, tenemos dos interfaces de aplicaciones (APIs):

- Open Database Connectivity (ODBC), utilizada por Microsoft (también se puede usar con Java).
- Java Database Connectivity (JDBC), desarrollada por Oracle para programas escritos en Java.

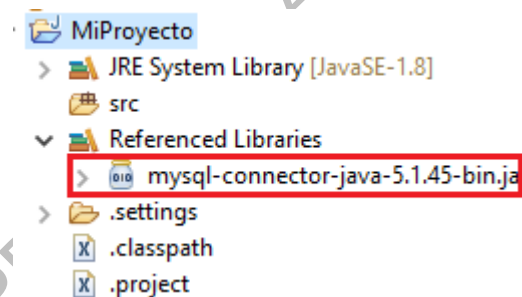
Ambas APIs nos dan las herramientas para conectar con una base de datos y recuperar y manipular la información que contiene.

Para poder establecer conexión con una base de datos en primer lugar vamos a necesitar descargarnos el driver JDBC para la BDD con la que vamos a trabajar. En el apartado de recursos y enlaces puedes encontrar el link a la página de descarga. Obtendrás un archivo comprimido que contendrá el conector JDBC. Debes incluir en tu proyecto el archivo cuyo nombre contenga "bin.jar".

Añadir esta librería a tu proyecto en Eclipse se hace clicando con el botón secundario del ratón sobre la carpeta del proyecto en la ventana del explorador de Eclipse. En el menú emergente, selecciona la opción "Build Path -> Configure build path...". Verás el siguiente cuadro de diálogo:



Añade la librería JDBC a tu proyecto desde el botón “Add External JARs...” y seleccionando el archivo “bin.jar” que has descomprimido. Verás que, en tu proyecto, aparecerá la librería JDBC:





14.2. Conexión con la base de datos

Antes de empezar a ver código, cabe destacar que crearemos una clase específica para trabajar con la base de datos. La llamaremos “GestionBDD”. En ella incluiremos todo el código que veremos a continuación, que se encargará de realizar las operaciones sobre la base de datos, separándolo así de la lógica y la vista de nuestra aplicación.

En primer lugar, declaramos unos atributos estáticos de la clase para guardar los datos de conexión: servidor donde se encuentra la base de datos, el nombre de ésta, usuario y contraseña para conectarnos al servidor. Además, crearemos también un atributo de tipo “Connection”, que se encarga de guardar los datos de la conexión y, a través de él, podremos ejecutar sentencias sobre la base de datos:

```
public class GestionBDD {  
    private static String datosConexion = "jdbc:mysql://localhost/";  
    private static String baseDatos = "miBDD";  
    private static String usuario = "root";  
    private static String password = "12345";  
    private Connection con;
```

Ejemplo de datos de conexión a una base de datos

Se puede apreciar que vamos a trabajar con la base de datos “miBDD” ubicada en la misma máquina que el programa que ejecutamos (“localhost”). Conectamos con MySQL con el usuario “root” y la contraseña “12345”. Se podría especificar también el puerto de conexión a la base de datos. Como estamos usando el puerto por defecto, que es el 3306, no hay que especificarlo, pero si usásemos otros, sí que tendríamos que ponerlo: “jdbc:mysql://localhost:3306/”.



Vamos a crear el constructor de nuestra clase “GestionBDD”, en el que estableceremos la conexión con la base de datos con los datos contenidos en los atributos de la clase:

```
public GestionBDD(){
    try {
        con = DriverManager.getConnection(datosConexion+"?useSSL=true", usuario,
password);
        try {
            //CREO LA BASE DE DATOS SI NO EXISTE
            crearBDD();
            //CREO LAS TABLAS SI NO EXISTEN
            crearTablas();
        } catch (Exception e) {
            e.printStackTrace();
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Ejemplo de método que conecta con una base de datos

Para establecer la conexión con la base de datos usamos el método “getConnection” de la clase DriverManager. Este método nos devuelve un objeto de la interfaz Connection. Se ha añadido al String de conexión el texto “useSSL=true”, para establecer una conexión segura.

El método “getConnection” recibe tres parámetros de tipo String que indican:

- URL de la base de datos a la que se quiere conectar.
- Usuario con el que se conecta.
- Password del usuario.

Obsérvese que hemos establecido una conexión con el servidor de bases de datos, pero aún no se ha especificado con qué base de datos vamos a trabajar. Esto se ha hecho así para aprovechar y crear la base de datos y sus tablas desde el código Java. Para ello llamamos a los métodos “crearBDD()” y “crearTablas()”.

Otra cosa a notar del código anterior, es que se hace un catch para excepciones de tipo SQLException. Este tipo de excepciones cubren los posibles errores con una base de datos, como la conexión o la ejecución de sentencias con errores.



14.3. Ejecutando sentencias en la base de datos

Seguimos implementando nuestra clase “GestionBDD”. Nos hemos quedado en la llamada a los métodos “crearBDD()” y “crearTablas()”. En ellos vamos a tener que ejecutar código en nuestra base de datos.

Para este propósito existe la clase “Statement”. Será la que usemos para poder ejecutar sentencias SQL, como INSERT, UPDATE, DELETE, SELECT, o CREATE como haremos a continuación. Creamos la base de datos:

```
private void crearBDD() throws Exception{
    String query = "create database if not exists "+ baseDatos +";";
    Statement stmt = null;
    try{
        stmt = con.createStatement();
        stmt.executeUpdate(query);
        con = DriverManager.getConnection(datosConexion+baseDatos+"?useSSL=true",
        usuario, password);
    }catch (SQLException e){
        e.printStackTrace();
    }finally{
        stmt.close();
    }
}
```

Ejemplo de método que crea una base de datos

Vemos que creamos un String que contendrá el código SQL a ejecutar. Se crea un objeto Statement con el constructor “createStatement” y luego llamamos al método “executeUpdate” pasándole como parámetro el String con el código SQL.

Se puede observar que se ha vuelto a establecer una conexión con la base de datos o, mejor dicho, se ha sustituido la conexión anterior por una nueva, en la que especificamos la base de datos con la que vamos a trabajar (ahora ya la tenemos creada, debemos seleccionarla).

En el bloque finally hacemos una llamada al método “close” del objeto Statement, liberando los recursos utilizados para ejecutar el código SQL en la base de datos. Es una buena práctica hacer esto cada vez que terminamos de usar un Statement.



Para la creación de las tablas en la base de datos podemos hacer lo siguiente:

```
private void crearTablas() throws Exception {  
    String query = "create table if not exists Tabla1(" +  
        "id int primary key auto_increment, " +  
        "campo1 varchar(100), " +  
        "campo2 int);";  
  
    Statement stmt = null;  
    try{  
        stmt = con.createStatement();  
        stmt.executeUpdate(query);  
    }catch (SQLException e){  
        e.printStackTrace();  
    }finally{  
        stmt.close();  
    }  
}
```

Ejemplo de método que crea tablas en una base de datos.

El proceso es el mismo que el usado para crear la base de datos.

14.3.1. Inserción, actualización y borrado de datos

Las operaciones INSERT, UPDATE y DELETE de SQL las ejecutamos de la misma forma desde Java. Vamos a ver un ejemplo de INSERT. El código de las otras dos operaciones sería igual, cambiando simplemente la sentencia SQL (y quizás un poco la cabecera del método, cambiando los parámetros de entrada en función de lo que queramos hacer).

Para poder ejecutar la sentencia INSERT con el objeto "Statement" usamos, de nuevo, el método "executeUpdate":



```
public void insertarDatos(String valor1, int valor2) throws Exception {  
    // Función que inserta un nuevo director  
    String query = "insert into miTabla(campo1, campo2) values('" + valor1 + "', " + valor2 + ");";  
    Statement stmt = null;  
    try {  
        stmt = con.createStatement();  
        stmt.executeUpdate(query);  
    } catch (SQLException e) {  
        e.printStackTrace();  
    }  
    finally{  
        stmt.close();  
    }  
}
```

Ejemplo de método que inserta datos en una base de datos

El método anterior recibe como parámetros los valores que se quieren insertar en la base de datos, que son una cadena de caracteres y un entero.

Cuando trabajamos con SQL, es importante el tipo de datos que usamos en una sentencia. Hay que recordar que las cadenas de caracteres, varchar, van entre comillas simples. Por tanto, en la sentencia INSERT que se hace, valor1 hay que ponerlo entre comillas simples. Un error muy común cuando ejecutamos sentencias SQL desde Java es olvidarse de las comillas simples en los campos que trabajan con el tipo varchar. En cambio, valor2, al tratarse de un entero, no va entre comillas simples.

Para ejecutar una sentencia INSERT usamos el método `executeUpdate` del objeto `Statement`.

14.3.2. Consultas en la base de datos: SELECT

Para ejecutar un SELECT sobre la BDD usaremos el método `executeQuery(String)` de la clase `Statement`. Todos los datos resultantes de la ejecución del SELECT se almacenan en un `ResultSet`.

El `ResultSet` contiene varias filas con el resultado de la consulta. Para movernos entre ellas usaremos un cursor, que inicialmente se sitúa delante de la primera fila. En la siguiente tabla se pueden ver algunos métodos de la clase `ResultSet`:



next()	Desplaza el cursor a la siguiente fila. Devuelve true si hay fila, false si no.
first()	Sitúa el cursor en la primera fila. Devuelve true si hay al menos una fila.
beforeFirst()	Sitúa el cursor delante de la primera fila.
previous()	Sitúa el cursor en la fila anterior, si hay. Si no devuelve false.
afterLast()	Sitúa el cursor detrás de la última fila.
absolute(int i)	Sitúa el cursor en la fila i. Devuelve true si hay una fila en esa posición. Las filas empiezan a numerarse a partir de 1.
relative(int i)	Desplaza el cursor y posiciones desde donde se encuentra actualmente. Devuelve true si hay una fila en esa posición.
last()	Sitúa el cursor en la última fila. Devuelve true si hay al menos una fila.

Existen diferentes tipos de ResultSet en función de sus características. Un criterio para clasificar un ResultSet es cómo podemos movernos por él:

- **TYPE_FORWARD_ONLY**: es la opción por defecto. El movimiento por sus registros es unidireccional hacia delante (de la primera fila hasta la última).
- **TYPE_SCROLL_INSENSITIVE**: podemos movernos hacia delante y hacia detrás. El ResultSet contiene los datos que tenía cuando se ejecutó el comando SQL, así que no reflejará posibles cambios posteriores en los datos de la base de datos.
- **TYPE_SCROLL_SENSITIVE**: igual que el anterior, pero si hubiera algún cambio en los datos de la base de datos después del comando SQL ejecutado, el ResultSet “se enterará” de este cambio. Por tanto, trabaja siempre con los datos más actuales de la base de datos.

Otro criterio de clasificación de los ResultSet es si sus datos son actualizables en la base de datos o no. A este criterio le llamamos concurrencia, y hay dos tipos:

- **CONCUR_READ_ONLY**: es la opción por defecto. Sus datos no pueden ser modificados y, por tanto, no se replican en la base de datos.
- **CONCUR_UPDATABLE**: los datos del ResultSet pueden ser actualizados y, los cambios, se actualizarán también en la base de datos.

Para acceder a los datos de cada fila de un ResultSet, usamos los métodos “get” (getInt, getBoolean...). A cada dato se puede acceder usando su posición en la fila (empezando por 1) o su nombre.



Veamos un ejemplo de método que realiza un SELECT sobre la tabla “miTabla” de la base de datos “miBDD”:

```
public ArrayList<String> listarDatos() throws Exception {  
    // Funcion que devuelve una lista con los datos de “miTabla”  
    ArrayList<String> lista = new ArrayList<String>();  
    String query = "select * from miTabla;";  
    Statement stmt = null;  
    ResultSet rs = null;  
    try {  
        stmt = con.createStatement();  
        rs = stmt.executeQuery(query);  
        while(rs.next()){  
            lista.add(rs.getInt(1)+ " - " + rs.getString(2) + " " + rs.getInt(3));  
        }  
    } catch (SQLException e) {  
        e.printStackTrace();  
    }  
    finally{  
        rs.close();  
        stmt.close();  
    }  
    return lista;  
}
```

Ejemplo de método que devuelve los datos contenidos en una tabla de una base de datos

El método que se ha implementado hace un SELECT sobre la base de datos y devuelve todos los datos de la tabla “miTabla”. Estos datos los añade, fila por fila, en un ArrayList, que será retornado al final del método. Como se ha dicho con anterioridad, todo el código que estamos viendo lo estamos implementando en la clase “GestionBDD”, con el objetivo de separar todo lo referente a la base de datos del resto de código de nuestra aplicación. Por tanto, el objetivo de que este método retorne un ArrayList con los datos de la base de datos, es poder usarlo en la lógica de la aplicación, en otra clase.

Vemos que para ejecutar una sentencia SELECT usamos el método executeQuery del objeto Statement. Este método devuelve un ResultSet. Para recuperar los datos que contiene, debemos recorrerlo fila por fila. Usamos un bucle while con la condición rs.next(). Como hemos visto, el método next mueve el cursor a la siguiente fila del ResultSet. Devuelve cierto si hay fila, false sino.



Para recuperar cada campo de una fila, se han usado los métodos `getInt` y `getString`. En “miTabla” recordemos que tenemos tres campos: `id` (entero), `campo1` (cadena de caracteres) y `campo2` (entero). Vemos que usamos el método `get` según el tipo de dato del campo al que vamos a acceder, y usamos el lugar que ocupa el campo en la tabla, empezando por 1. Un error muy común es empezar por la posición 0, como hacemos para acceder a la primera posición de un array, `ArrayList`, etc.

De igual forma que, al final de nuestro código, llamamos al método `close` de la clase `Statement` para liberar recursos que ya no usamos, hacemos lo propio con el `ResultSet`.

14.3.3. Ejemplo de uso de JDBC y Swing

En el siguiente vídeo puedes ver un ejemplo de aplicación que usa un base de datos MySQL e interfaz gráfica:



UF6_14_3_3_Ejemplo de uso de JDBC y Swing



14.3.4. Inserción, actualización y borrado de datos usando ResultSet

El ResultSet no lo usamos exclusivamente para hacer un SELECT sobre la base de datos. Podemos usarlo para las operaciones INSERT, UPDATE y DELETE, aunque la forma más habitual de ejecutar estas operaciones es mediante el objeto Statement, como hemos visto.

Vamos a ver ejemplo de código en el que se usa la clase ResultSet para ejecutar un INSERT:

```
Connection con = DriverManager.getConnection(url, user, password);
Statement stmt = con.createStatement(ResultSet.TYPE_FORWARD_ONLY,
                                     ResultSet.CONCUR_UPDATABLE);
ResultSet rs = stmt.executeQuery("SELECT * FROM usuario");
rs.moveToInsertRow();
rs.updateInt("id", 47);
rs.updateString("nombre", "Dani");
rs.updateString("contrasena", "miPass");
rs.insertRow()
```

Ejemplo de inserción de datos usando ResultSet

Vamos a comentar el código anterior.

En primer lugar, obsérvese cómo se ha creado el objeto de la clase "Statement", pasándole dos parámetros:

- TYPE_FORWARD_ONLY: a la hora de acceder a los datos de un ResultSet sólo se podrán recorrer sus filas secuencialmente empezando por la primera.
- CONCUR_UPDATABLE: los datos del ResultSet se pueden modificar. Los cambios afectarán también a la base de datos.

El método moveToInsertRow hace que el cursor del ResultSet se sitúe en una fila especial para las inserciones. Es un buffer en el que se guardará la nueva fila a insertar en la base de datos.

A continuación, se van añadiendo valores a la fila a insertar mediante los métodos updateInt, updateString, etc. Con estos métodos damos el valor que debe tener cada campo, teniendo en cuenta su nombre en la tabla y el tipo de dato, claro.

Por último, para ejecutar el INSERT y guardar la nueva fila en la tabla, se ejecuta el método insertRow.



Veamos un ejemplo de UPDATE con ResultSet. Se va a modificar el nombre del usuario cuyo id es 46:

```
Connection con = DriverManager.getConnection(url, user, password);
Statement stmt = con.createStatement(ResultSet.TYPE_FORWARD_ONLY,
                                     ResultSet.CONCUR_UPDATABLE);
ResultSet rs = stmt.executeQuery("SELECT * FROM usuario");
boolean encontrado = false;
while(rs.next() && !encontrado){
    if(rs.getInt("id") == 46){
        rs.updateString("nombre", "Helena");
        rs.updateRow();
        encontrado = true;
    }
}
```

Ejemplo de modificación de datos usando ResultSet

Y, ahora, un ejemplo de DELETE. Borramos el usuario con id igual a 46:

```
Connection con = DriverManager.getConnection(url, user, password);
Statement stmt = con.createStatement(ResultSet.TYPE_FORWARD_ONLY,
                                     ResultSet.CONCUR_UPDATABLE);
ResultSet rs = stmt.executeQuery("SELECT * FROM usuario");
boolean encontrado = false;
while(rs.next() && !encontrado){
    if(rs.getInt("id") == 46){
        rs.deleteRow();
        encontrado = true;
    }
}
```

Ejemplo de borrado de datos usando ResultSet



14.3.5. PreparedStatement

Las operaciones de inserción, modificación, borrado y consulta sobre la base de datos las hemos hecho usando la clase Statement.

Existe también la clase PreparedStatement, que hereda de Statement. Con PreparedStatement también podemos ejecutar comandos SQL. La diferencia con su clase madre es que los posibles parámetros de entrada son precompilados, con lo que la ejecución de la sentencia será más rápida.

Su uso es muy adecuado cuando se va a ejecutar la misma sentencia SQL muchas veces.

Para usar PreparedStatement debemos habilitar esta propiedad cuando establecemos la conexión con la base de datos:

```
Connection con =  
DriverManager.getConnection("jdbc:mysql://server/bdd?useServerPrepStmts=true", user, pass);
```

Habilitando PreparedStatement

Ahora podemos ejecutar cualquier operación SQL usando la siguiente estructura:

```
PreparedStatement psInsert = con.prepareStatement("insert into usuario values (?, ?, ?)");  
psInsert.setInt(1, 46);  
psInsert.setString(2, "Dani");  
psInsert.setString(3, "MiPass");  
psInsert.executeUpdate();  
...  
psInsert.setInt(1, 67);  
psInsert.setString(2, "Raul");  
psInsert.setString(3, "MiPass2");  
psInsert.executeUpdate();  
psInsert.close();
```

Ejemplo de INSERT con PreparedStatement.

Hemos puesto interrogantes allá donde irán los datos a insertar. Cada interrogante se identifica luego con un número, empezando por 1, y con los métodos set rellenamos los valores. Finalmente ejecutamos el INSERT en la base de datos con el método executeUpdate.

Como vemos, podemos ejecutar varias instrucciones de inserción con un único PreparedStatement.



14.4. Transacciones

Una transacción es un conjunto de sentencias que deben ejecutarse como un todo. Es decir, o se ejecutan todas o ninguna.

El típico ejemplo es de una transacción entre dos cuentas bancarias. El proceso, pensando en operaciones de SQL, sería el siguiente: hacemos un UPDATE sobre la primera cuenta, restándole saldo, y a continuación otro UPDATE sobre la segunda cuenta, sumándole saldo. ¿Qué pasaría si se produjera algún problema en el segundo UPDATE, cuando el primero ya se ha ejecutado? Vemos que, en este caso, las dos instrucciones deben ejecutarse en conjunto. Si la segunda fallara, debería deshacerse la primera.

En los ejemplos que hemos visto de inserción, modificación y borrado de datos, estas instrucciones producían cambios en la base de datos en cuanto se ejecutaban. Esto es así porque la propiedad auto-commit de la base de datos está habilitada.

Cuando queramos trabajar con transacciones, debemos quitar el modo auto-commit.



Hagamos el ejemplo de la transacción bancaria con código:

```
Connection con = DriverManager.getConnection(url, user, password);
String query1 = "update cuenta set saldo = saldo - 500 where num_cuenta = 1";
String query2 = "update cuenta set saldo = saldo + 500 where num_cuenta = 2";
Statement stmt = null;
try{
    con.setAutoCommit(false);
    stmt = con.createStatement();
    stmt.executeUpdate(query1);
    stmt.executeUpdate(query2);
} catch(SQLException e){
    e.printStackTrace();
    if(con != null) {
        try{
            System.out.println("Roll back de la transacción");
            con.rollback();
        } catch (SQLException e2){
            e2.printStackTrace();
        }
    }
} finally {
    stmt.close();
    con.setAutoCommit(true);
}
```

Ejemplo de ejecución de transacciones

Si alguno de los UPDATE que ejecutamos falla, se producirá una excepción, pero la conexión seguirá vigente (por tanto, no será nula). Llamamos al método rollback para deshacer los cambios producidos en el primer UPDATE en caso de haberse ejecutado.



Recursos y enlaces

- Descarga del driver JDBC para MySQL:
<https://dev.mysql.com/downloads/connector/j/5.1.html>



- Documentación oficial JDBC:
<https://docs.oracle.com/javase/tutorial/jdbc/basics/index.html>



Conceptos clave

- **JDBC**: API que nos proporciona las herramientas necesarias para trabajar con bases de datos usando el lenguaje Java.
- **Statement**: clase que se usa para ejecutar sentencias SQL.
- **ResultSet**: clase que se usa como contenedor de los resultados de una consulta SQL y proporciona acceso a los datos por filas.
- **Transacción**: conjunto de sentencias que deben ejecutarse como un todo. Es decir, o se ejecutan todas o ninguna.



Test de autoevaluación

¿Qué clase es la encargada de enviar sentencias SQL a la base de datos?

- a) Statement
- b) JDBC
- c) DriverManager
- d) Connection

¿Qué método emplearé si quiero ejecutar una sentencia INSERT en la base de datos?

- a) executeInsert
- b) executeUpdate
- c) executeQuery
- d) executeSQL

Si un SELECT a una base de datos devuelve algo, ¿en qué tipo de variable se guarda el resultado?

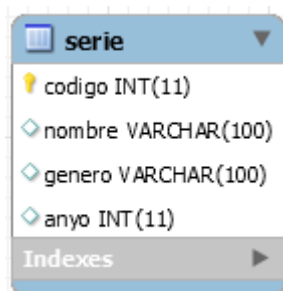
- a) Statement
- b) ResultSet
- c) Connection
- d) DriverManager



Ponlo en práctica

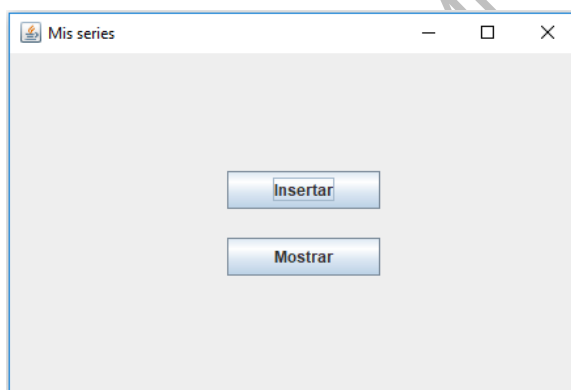
Actividad 1

Crea una aplicación en Java que gestione la información de las series almacenadas en una base de datos. Únicamente se dispone de la siguiente tabla:

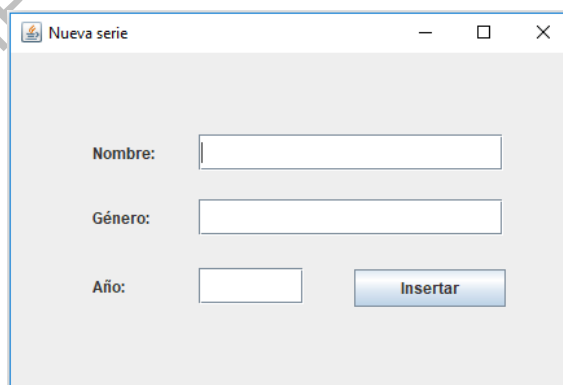


serie	
codigo	INT(11)
nombre	VARCHAR(100)
genero	VARCHAR(100)
año	INT(11)
Indexes	

La aplicación debe tener interfaz gráfica con una pantalla principal como la que sigue:



Al clicar sobre el botón "Insertar" se abrirá una nueva ventana:



El usuario introducirá los datos de la nueva serie y, al clicar sobre el botón "Insertar", se creará un nuevo registro en la base de datos, se cerrará la ventana actual, y se volverá a mostrar la pantalla inicial.



Al clicar sobre el botón “Mostrar” se abrirá la siguiente pantalla:

The screenshot shows a window titled "Mostrar series" with a standard Windows-style title bar (minimize, maximize, close buttons). Inside the window, there are three labels with corresponding input fields: "Nombre:" with a text box containing "q", "Género:" with a text box containing "q", and "Año:" with a text box containing "1". At the bottom of the window, there are two buttons: "<< Anterior" on the left and "Siguiete >>" on the right. The text "Siguiete" appears to be a typo for "Siguiente".

En esta ventana se mostrarán, una a una, las series que tenemos en la base de datos. Al cargar la página, se mostrará la primera serie. El botón “Anterior” estará deshabilitado ya que estamos mostrando la primera serie (no hay anteriores). Se tendrá que habilitar cuando no estemos mostrando la primera serie. De igual forma, el botón “Siguiete” se tendrá que deshabilitar cuando estemos mostrando la última serie.



SOLUCIONARIOS

Test de autoevaluación

¿Qué clase es la encargada de enviar sentencias SQL a la base de datos?

- a) **Statement**
- b) JDBC
- c) DriverManager
- d) Connection

¿Qué método emplearé si quiero ejecutar una sentencia INSERT en la base de datos?

- a) executeInsert
- b) **executeUpdate**
- c) executeQuery
- d) executeSQL

Si un SELECT a una base de datos devuelve algo, ¿en qué tipo de variable se guarda el resultado?

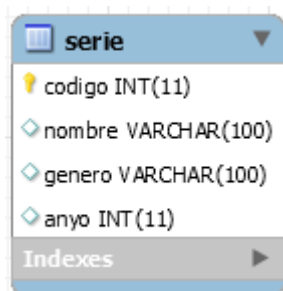
- a) Statement
- b) **ResultSet**
- c) Connection
- d) DriverManager



Ponlo en práctica

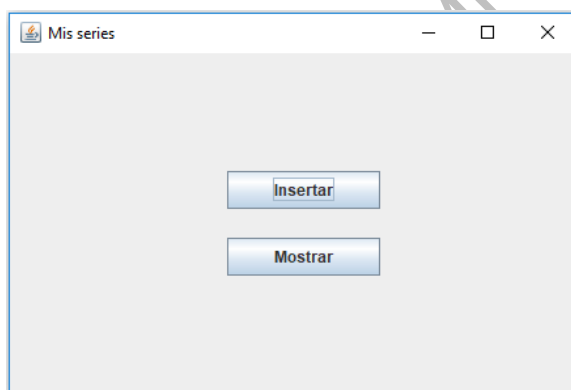
Actividad 1

Crea una aplicación en Java que gestione la información de las series almacenadas en una base de datos. Únicamente se dispone de la siguiente tabla:

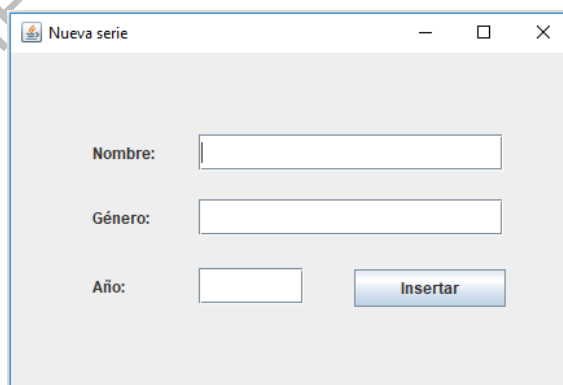


serie	
codigo	INT(11)
nombre	VARCHAR(100)
genero	VARCHAR(100)
año	INT(11)
Indexes	

La aplicación debe tener interfaz gráfica con una pantalla principal como la que sigue:



Al clicar sobre el botón “Insertar” se abrirá una nueva ventana:



El usuario introducirá los datos de la nueva serie y, al clicar sobre el botón “Insertar”, se creará un nuevo registro en la base de datos, se cerrará la ventana actual, y se volverá a mostrar la pantalla inicial.



Al clicar sobre el botón “Mostrar” se abrirá la siguiente pantalla:

The screenshot shows a window titled "Mostrar series" with a standard Windows-style title bar (minimize, maximize, close buttons). Inside the window, there are three labels with corresponding text input fields: "Nombre:" with the value "q", "Género:" with the value "q", and "Año:" with the value "1". At the bottom of the window, there are two buttons: "<< Anterior" on the left and "Siguiete >>" on the right. The text "Siguiete" appears to be a typo for "Siguiente".

En esta ventana se mostrarán, una a una, las series que tenemos en la base de datos. Al cargar la página, se mostrará la primera serie. El botón “Anterior” estará deshabilitado ya que estamos mostrando la primera serie (no hay anteriores). Se tendrá que habilitar cuando no estemos mostrando la primera serie. De igual forma, el botón “Siguiete” se tendrá que deshabilitar cuando estemos mostrando la última serie.

El solucionario está disponible en la versión interactiva del aula.