# TEMA 11. CONTROL DE ACCESO Y MANTENIMIENTO DE BASES DE DATOS RELACIONALES

## Índice



## Bases de Datos Relacionales

Muchas de las aplicaciones que se desarrollan hoy en día incluyen el trabajo con bases de datos.

Las bases de datos más usadas en la actualidad son las bases de datos relacionales.

Uno de los Sistemas Gestores de Bases de Datos más usados en la actualidad es MySQL. Es por ello que lo vamos a usar para el desarrollo de aplicaciones Java con acceso a bases de datos.

## Creación de una Base de Datos Relacional Sencilla

Para poder crear una base de datos MySQL debemos tener instalado un servidor HTTP configurado para usar MySQL. El modo más sencillo de instalar un servidor HTTP configurado para usar MySQL consiste en instalar XAMPP.

## Instalar Xampp en Windows

Vamos a instalar la versión la versión instalable de XAMPP (xampp-windows-x64-8.1.5-0-VS16-installer.exe en mi caso). Para ello basta con descomprimir el contenido en la carpeta **c:\xampp**.

La ruta de XAMPP debe de ser **c:\xampp**

Las páginas web deben estar en **c:\xampp\htdocs** que es el directorio por defecto.

Una vez instalado XAMPP debemos ir al **panel de control de XAMPP** e **iniciar** el servidor **Apache**.

Si hay algún problema con el puerto por defecto de **HTTP** (**puerto 80**) debemos editar el fichero de configuración **c:\xampp\apache\conf\httpd.conf** y sustituir el puerto de escucha (linea **Listen 80**) por otro que esté disponible (por ejemplo reemplazar la línea por **Listen 8088**)

Si hay algún problema con el puerto por defecto de **HTTPS** (**puerto 443**) debemos editar el fichero de configuración **c:\xampp\apache\conf\extra\httpd-ssl.conf** y sustituir el puerto de escucha (linea **Listen 443**) por otro que esté disponible (por ejemplo reemplazar la línea por **Listen 4433**). También debemos cambiar el valor 443 por 4433 en **<VirtualHost _default_:4433>** y añadir un nuevo servidor con los datos **ServerName localhost:4433**

El archivo ejecutable del **panel de control de XAMPP** es **c:\xampp\xampp-control.exe**. Podemos crear un **Acceso Directo** en el Escritorio para facilitar la ejecución.

Abrimos el panel de control y pulsamos **Start** en la linea de **Apache** y en la linea **MySQL**. Si todo ha ido bien ya tendremos el servidor funcionando.

Una vez que hemos comprobado que el servidor Apache está en ejecución vamos al navegador e introducimos la dirección **http://localhost/** para comprobar que funciona correctamente. Esto es válido si el **puerto** de **HTTP** es el **80**, si hemos cambiado el **puerto** de HTTP por el **8088** la dirección sería la **http://localhost:8088/**

Para probar el servicio https vamos al navegador e introducimos la dirección **https://localhost/**. Esto es válido si el **puerto** de **HTTPS** es el **443**, si hemos cambiado el **puerto** de HTTPS por el **4433** la dirección sería la **https://localhost:4433/**

Para modificar el sitio web por defecto debemos de ir a la carpeta **c:\xampp\htdocs** y copiar allí los documentos de nuestro sitio web.

**Nota**: Si XAMPP no funciona correctamente debemos borrar la carpeta c:\xampp y descargar e instalar la versión instalable de XAMPP.

## MySQL - Configuración

Para modificar la configuración de MySQL podemos editar el archivo de configuración **my.ini** que se encuentra en **C:\xampp\mysql\bin**.

Esto es útil porque en las últimas versiones de Java para poder establecer una conexión con una base de datos MySQL es obligatorio que la zona horaria de MySQL esté definida.

Para establecer la zona horaria de MySQL editamos el archivo **my.ini**, vamos a la sección [mysqld] y añadimos al final la línea *default-time-zone='+01:00'*

```
[mysqld]
…
default-time-zone='+01:00'
```

También podemos modificar MySQL usando una interfaz web. Para ello, una vez instalado XAMPP e iniciados Apache (con HTTP en el puerto 80) y MySQL sin problemas vamos al enlace http://localhost/phpmyadmin/.

Una vez allí hacemos clic sobre la opción **Nueva** para crear una nueva base de datos de nombre **bdalumnos** con cotejamiento **utf8_spanish_ci**.

Una vez creada la base de datos bdalumnos vamos a crear tres tablas de nombre alumnos, asignaturas, y calificaciones que tienen las siguientes características

**alumnos**
dni. Campo de tipo texto de 9 caracteres. Clave primaria.
nombre. Campo de tipo texto de 30 caracteres.
apellidos. Campo de tipo texto de 50 caracteres.
grupo. Campo de tipo texto de 4 caracteres.

**asignaturas**
codasignatura. Campo de tipo texto de 4 caracteres. Clave primaria
nombreasignatura. Campo de tipo texto de 30 caracteres.
descripcion. Campo de tipo texto de 200 caracteres.

**calificaciones**
dni. Campo de tipo texto de 9 caracteres. Clave ajena.
codasignatura. Campo de tipo texto de 4 caracteres. Clave ajena.
nota. Campo de tipo entero sin signo de tamaño 2.

Podemos crear un **archivo de texto** en formato **UTF-8** con las instrucciones SQL necesarias para crear las tablas con datos de prueba e importar el archivo desde la opción **importar** o **copiar todas instrucciones** a ejecutar en la pestaña **SQL**.

Yo he creado un documento de texto de nombre bdalumnos.txt con el siguiente contenido:

```
create table bdalumnos.alumnos(
 dni varchar(9) primary key,
 nombre varchar(30) not null,
 apellidos varchar(50) not null,
 grupo varchar(4) not null
);

insert into bdalumnos.alumnos values ('11111111A','N1','A1','1AS3');
insert into bdalumnos.alumnos values ('22222222B','N2','A2','1DW3');
insert into bdalumnos.alumnos values ('33333333C','N3','A3','2AS3');
insert into bdalumnos.alumnos values ('44444444D','N4','A4','1DW3');
insert into bdalumnos.alumnos values ('55555555E','N5','A5','1AS3');
insert into bdalumnos.alumnos values ('66666666F','N6','A6','1AS3');
insert into bdalumnos.alumnos values ('77777777G','N7','A7','2DW3');
insert into bdalumnos.alumnos values ('88888888H','N8','A8','1AS3');

create  table bdalumnos.asignaturas(
 codasignatura varchar(4) primary key,
 nombreasignatura varchar(30) not null,
 descripcion varchar(200)
);

insert into bdalumnos.asignaturas values ('BD','Bases de Datos','');
insert into bdalumnos.asignaturas values ('ED','Entornos de Desarrollo','');
insert into bdalumnos.asignaturas values ('FOL','Formación y Orientación Laboral','');
insert into bdalumnos.asignaturas values ('ISO','Implantación de Sistemas Operativos','');
insert into bdalumnos.asignaturas values ('PROG','Programación','');

create table bdalumnos.calificaciones(
 dni varchar(9) not null,
 codasignatura varchar(4) not null,
 nota int(2) unsigned not null,
 foreign key (dni) references alumnos(dni) on delete cascade,
 foreign key (codasignatura) references asignaturas(codasignatura) on delete cascade,
 unique (dni,codasignatura)
);

insert into bdalumnos.calificaciones values ('11111111A','BD',5);
insert into bdalumnos.calificaciones values ('11111111A','ED',4);
insert into bdalumnos.calificaciones values ('11111111A','FOL',7);
insert into bdalumnos.calificaciones values ('11111111A','PROG',8);
insert into bdalumnos.calificaciones values ('22222222B','BD',5);
insert into bdalumnos.calificaciones values ('22222222B','ED',4);
insert into bdalumnos.calificaciones values ('33333333C','FOL',7);
insert into bdalumnos.calificaciones values ('77777777G','PROG',8);
```

## Conexiones con Bases de Datos Relacionales

Para que una aplicación Java se pueda comunicar con una base de datos MySQL debemos instalar un driver. El driver JDBC para permitir conexiones a bases de datos MySQL desde aplicaciones lo podemos descargar en el siguiente enlace http://dev.mysql.com/downloads/connector/j/5.0.html. Seleccionamos la opción **Platform Independent** y descargamos el archivo ZIP.

Una vez descargado, descomprimimos el archivo. De todos los ficheros descomprimidos **solo** nos interesa **el archivo .jar** (**mysql-connector-java-8.0.x.jar**) que es el archivo que contiene la clase Driver que necesitamos.

Para poderlo usar dentro de nuestro proyecto en Eclipse debemos importar el archivo .jar a nuestro proyecto.

Vamos a crear un nuevo paquete de nombre **basesdedatos** dentro de nuestro proyecto. Una vez creado vamos a crear dentro la clase Java **BDConexion** que se intentará conectar a la base de datos MySQL de nombre **bdalumnos** y mostrará un mensaje indicando si se ha realizado la conexión correctamente o no.

Copiamos el archivo .jar con el **Driver** dentro del paquete de nombre **basesdedatos**. Hacemos clic sobre él con el botón derecho del ratón y en el menú desplegable seleccionamos la opción **Build Path** -> **Add to Build Path**

A partir de este momento ya podemos usar el Driver en nuestro proyecto.

Para poder usar las clases que nos permiten conectarnos a una base de datos MySQL debemos importar las clases **java.sql.Connection**, **java.sql.DriverManager**, y **java.sql.SQLException**;

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
```

Una vez importadas las clases necesarias, podemos proceder a la conexión mediante **DriverManager**.

Por ejemplo, para **conectarnos a** la base de datos MySQL de nombre **bdalumnos** que se encuentra en el equipo, con el **usuario root** y **contraseña ""**, escribimos

```
Connection conexion = DriverManager.getConnection("jdbc:mysql://localhost/bdalumnos", "root", "");
```

Si todo ha ido bien la conexión se habrá realizado correctamente.

Como las operaciones con bases de datos son propensas a errores se deben controlar las **excepciones**. Las excepciones típicas de bases de datos son las **SQLException**.

Para controlar errores en la aplicación anterior escribimos

```
try{
  Connection conexion = DriverManager.getConnection("jdbc:mysql://localhost/bdalumnos", "root", "");
 // si se ha conectado correctamente
 System.out.println("Conexión Correcta.");
 // cierro la conexion
 conexion.close();
}
catch (SQLException e){
 // si NO se ha conectado correctamente
 e.printStackTrace();
 System.out.println("Error de Conexión");
}
```

Si se produce un error MySQL debido a un problema con la zona horaria, debemos modificar la configuración de MySQL para añadir la configuración de la zona horaria (solución definitiva) o ir a phpMyAdmin y ejecutar la siguiente sentencia SQL (solución provisional)

```
SET GLOBAL time_zone = '+1:00';
```

## Herramientas de Manipulación de Datos SQL. Resulset

Una vez que nos hemos conectado a una base de datos, para enviar comandos SQL a la base de datos se usa la clase Java **java.sql.Statement**. Para obtener un objeto de esta clase a partir de la conexión escribimos

```
Statement st = conexion.createStatement();
```

En JDBC por defecto un ResultSet es TYPE_FORWARD_ONLY y CONCUR_READ_ONLY. Esto quiere decir que el ResultSet solo se puede mover hacia delante y solo puede leer los datos. No tiene ninguna capacidad para actualizarlos. Existen otros dos tipos de ResultSet: TYPE_SCROLL_INSENSITIVE y TYPE_SCROLL_SENSITIVE, el primero de ellos es capaz de mover el ResultSet hacia delante y hacia atrás.

Si tenemos problemas al trabajar con el ResultSet podemos modificar las opciones de creación del Statement. Por ejemplo, si al intentar usar **rs.first()** para **comprobar si hay registros** en el ResultSet se produce una SQLException del tipo ResultSet.TYPE_FORWARD_ONLY debemos modificar la creación del Statement de la siguiente manera

```
Statement st = conexion.createStatement( ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY);
```

Una vez que ya tenemos un objeto Statement asociado a una conexión, podemos ejecutar consultas. Los objetos Statement tienen el método **executeUpdate**() para ejecutar sentencias SQL que impliquen **modificaciones** en la base de datos (INSERT, UPDATE, DELETE, etc) y el método **executeQuery**() para **consultas** (SELECT y similares) que devuelven un conjunto de registros.

El método **executeQuery**() devuelve un objeto de tipo ResultSet que contiene todos los datos devueltos por la consulta. Para usar ResulSet debemos importar la clase **java.sql.ResultSet**.

Por ejemplo, si queremos realizar una consulta que obtenga los datos de todos los alumnos de la tabla alumnos escribimos.

```
ResultSet rs = st.executeQuery("SELECT * FROM bdalumnos.alumnos;");
```

Para obtener los datos de uno de los campos del registro actual del ResulSet podemos usar el método **getObject()** indicando como parámetro el nombre del campo del registro del que queremos obtener su valor entre comillas dobles **"**. Por ejemplo, para obtener el valor del campo dni del registro actual del ResulSet escribimos

```
rs.getObject("dni")
```

Para recorrer todos los registros de un ResulSet podemos usar el método **rs.next()**. Por ejemplo, para mostrar por pantalla los datos de todos los alumnos que hay en el ResultSet rs escribimos

```
while (rs.next()){
 System.out.println("DNI: " + rs.getObject("dni") + ", nombre: " + rs.getObject("nombre") + ", Apellidos: " + rs.getObject("apellidos") + ", Grupo: " + rs.getObject("grupo"));
}
```

Para comprobar si un ResulSet tiene registros podemos usar el método **rs.first()** que devuelve **true** si el ResulSet tiene registros o **false** si no tiene registros. El problema es que se posiciona ya en el primer registro y si usamos el método rs.next para recorrer el ResulSet se salta el primer registro. Para evitarlo antes de ejecutar el método rs.next debemos ejecutar el método **rs.beforeFirst()** que se posiciona antes del primer registro. Por ejemplo, para controlar si hay registros o no en la tabla alumnos y, si hay registros mostrarlos por pantalla escribimos

```
// compruebo si hay registros
if(rs.first()) {
 // si hay registros
 // vuelvo al primero
 rs.beforeFirst();
 // recorro registro a registro el ResultSet
 while (rs.next()){
   System.out.println("DNI: " + rs.getObject("dni") + ", nombre: " + rs.getObject("nombre") + ", Apellidos: " + rs.getObject("apellidos") + ", Grupo: " + rs.getObject("grupo"));
 }
}
else {
 // si no hay registros
 System.out.println("La tabla no tiene Registros");
}
```

Es muy importante que una vez que ya no vayamos a usar más el ResulSet lo cerremos para evitar problemas. Para cerrar un ResulSet escribimos

```
rs.close();
```

También es importante cerrar el Statement si vamos a cambiar de una sentencia de tipo executeQuery a una sentencia de tipo executeUpdate.

```
st.close();
```

Para **actualizar** los datos del alumno con dni '11111111A' y cambiar el valor del grupo a '1DW3' debemos usar el método **executeUpdate**() de la siguiente manera

```
// actualizo el valor del grupo del alumno '11111111A' a '1DW3'
st.executeUpdate("UPDATE bdalumnos.alumnos SET grupo='1DW3' WHERE dni='11111111A';");
```

Si queremos **borrar** el alumno con dni '11111111A' debemos usar el método **executeUpdate**() de la siguiente manera

```
// borro el alumno '11111111A'
st.executeUpdate("DELETE FROM bdalumnos.alumnos WHERE dni='11111111A';");
```

Si queremos **insertar** un alumno con dni '88888888H' debemos usar el método **executeUpdate**() de la siguiente manera

```
// inserto el alumno '99999999A'
st.executeUpdate("INSERT INTO bdalumnos.alumnos VALUES ('99999999A','N9','A9','2AS3'); ");
```

Cuando ejecutamos consultas de modificación o de borrado usando el método **executeUpdate**() debemos **comprobar** cuantos **registros** han sido **modificados** ya que las consultas de modificación o borrado se pueden ejecutar correctamente pero no afectar a ningún registro (porque no existe un registro con esos datos).

Por ejemplo, cuando intentamos borrar un registro que no está en la tabla o intentamos actualizar los datos de un registro que no está en la tabla la consulta se ejecuta correctamente pero no borra ni actualiza ningún registro.

Para controlar el número de registros modificados podemos usar una variable de tipo entero que recoja el valor devuelto por la ejecución de la consulta ejecutada en el método **executeUpdate**().

Por ejemplo, para controlar si el registro con código '11111111A' se ha borrado correctamente escribimos

```
int registrosmodificados;
registrosmodificados = st.executeUpdate("DELETE FROM bdalumnos.alumnos WHERE dni='11111111A';");
if (registrosmodificados > 0) {
 // si se ha borrado algun registro
 System.out.println("El registro ha sido borrado");
}
else {
 // si NO se ha borrado algun registro
 System.out.println("NO se ha borrado ningún registro");
}
```

Cuando se ejecuta una sentencia **INSERT** se puede producir una excepción SQL de clave duplicada. Para controlar esa excepción escribimos

```
try {
 …
 int registrosmodificados;
 registrosmodificados = st.executeUpdate("INSERT INTO bdalumnos.alumnos VALUES
('99999999A','N9','A9','2AS3');");
…
} catch (SQLException e) {
 // si se produce una excepción SQL
 int errorcode = e.getErrorCode();
 if (errorcode == 1062) {
  // si es un error de clave suplicada
  System.out.println("Error Clave Duplicada. Ya existe un registro con esa clave.");
 }
 else {
  System.out.println("Error SQL Numero "+e.getErrorCode()+":"+e.getMessage());
 }
```

```
}
```

Es muy importante que después de ejecutar una consulta **cerremos el Statement** para evitar problemas a la hora de realizar consultas posteriores.

```
st.close();
```

También es importante que cuando dejemos de trabajar con la base de datos cerremos la conexión mediante el comando

```
conexion.close();
```

Los objetos ResultSet son muy útiles para obtener datos. Sin embargo, el hecho de necesitar estar **conectado** el **ResultSet** al origen de datos para funcionar correctamente hace que trabajar con ResulSet no sea adecuado en muchas aplicaciones cliente / servidor. En estos casos es mejor trabajar con **RowSets**.

## Ejercicios de Manipulación de Datos SQL en Bases de Datos Relacionales

1. Crea la clase Java **BDSeleccionAlumno** que muestra por pantalla el contenido de la tabla alumnos de la base de datos MySQL bdalumnos. Si se ha producido algún error muestra un mensaje informativo. Para el proceso usa ResultSet. Si no hay registros muestra el mensaje “La tabla no tiene Registros”.

2. Crea la clase Java **BDSeleccionAlumnoHTML** que genera un documento HTML de nombre **alumnos.html** que muestra el contenido de la tabla alumnos de la base de datos MySQL bdalumnos en una tabla HTML. Si se ha producido algún error muestra un mensaje informativo. Para el proceso usa ResultSet. Si no hay registros muestra el mensaje “La tabla no tiene Registros”.

3. Crea la clase Java **BDUpdateAlumno** que modifica en la tabla alumnos de la base de datos MySQL bdalumnos los datos del alumno de dni '11111111A', para que el grupo sea '1DW3'. Si todo ha ido bien y se ha modificado algún registro muestra todo el contenido de la tabla alumnos por pantalla. Si se ha producido algún error muestra un mensaje informativo. Para el proceso usa ResultSet.

4. Crea la clase Java **BDDeleteAlumno** que borra de la tabla alumnos de la base de datos MySQL bdalumnos el alumno de dni '12345678A'. Si todo ha ido bien y se ha modificado algún registro muestra todo el contenido de la tabla alumnos por pantalla. Si se ha producido algún error muestra un mensaje informativo. Para el proceso usa ResultSet.

5. Crea la clase Java **BDInsertAlumno** que inserta en la tabla alumnos de la base de datos MySQL bdalumnos un alumno de dni '12345678A', nombre 'Nuevo', apellidos 'Alumno', y grupo '1AS3'. Si todo ha ido bien y se ha insertado algún registro muestra todo el contenido de la tabla alumnos por pantalla. Si se ha producido algún error muestra un mensaje informativo. Para el proceso usa ResultSet.

6. Crea la clase Java **ArrayListComplejosMenuBD** que modifica la clase ArrayListComplejosMenuSerializable para que al inicio de la ejecución se carguen los datos desde la tabla complejos de la base de datos bdcomplejos y para que al final de la ejecución, si los datos han sido modificados, se graben en esa misma tabla de la base de datos.

7. Crea la clase Java **ArrayListAlumnosMenuBD** que modifica la clase ArrayListComplejosMenuBD para que al inicio de la ejecución se carguen los datos desde la tabla alumnos de la base de datos bdalumnos y para que al final de la ejecución, si los datos han sido modificados, se graben en esa misma tabla de la base de datos.

## Componentes Gráficos para Trabajar con Bases de Datos. JTable

Debido a los problemas múltiples problemas que se producen al trabajar directamente con ResulSet y RowSet, normalmente la manipulación de los elementos de una base de datos se realiza a través de componentes gráficos. Aunque el principal problema de usar componentes gráficos intermedios es tener que actualizar los datos de la base de datos cuando se actualicen los datos del componente gráfico y viceversa.

Uno de los componentes que más se usa es **JTable**.

Para especificar las cabeceras de las columnas de una tabla podemos hacerlo de forma manual escribiendo nosotros los datos. Por ejemplo, para la tabla que muestra los datos de los alumnos vamos a crear un **Vector**

de datos de tipo **String** de nombre columnas que va a almacenar el valor de la cabecera de cada una de las columnas.

```
// cabeceras de las columnas
Vector<String> columnas = new Vector<String>();
columnas.add("DNI");
columnas.add("Nombre");
columnas.add("Apellidos");
columnas.add("Grupo");
```

También podemos usar un **ResultSetMetaData** de nombre **metaDatos** que guarde todos los metadatos del ResulSet y después sacar de él los nombres que tienen las columnas en la base de datos. Para ello escribimos.

```
// cabeceras de las columnas
ResultSetMetaData metaDatos = rs.getMetaData();
// Se obtiene el número de columnas.
int numeroColumnas = metaDatos.getColumnCount();

Vector<String> columnas = new Vector<String>();
// Se obtiene cada una de las etiquetas para cada columna
for (int i = 0; i < numeroColumnas; i++){
 // cojo el valor de la etiqueta de la columna
 // los índices del rs empiezan en 1 pero los índices de las columnas empiezan en 0
 columnas.add(metaDatos.getColumnLabel(i + 1));
}
```

Para cargar datos en una JTable podemos usar la clase Java Vector. Lo que haremos será cargar los datos de cada fila en un **Vector** de nombre **fila** y añadirlo a un **Vector de Vectores** de nombre **datosTabla**.

```
// creo el vector para los datos de la tabla
Vector<Vector<String>> datosTabla = new Vector<Vector<String>>();
// añado uno a uno los alumnos al vector de datos
Vector<String> fila;
while (rs.next()) {
 fila = new Vector<String>();
 fila.add(rs.getString("dni"));
 fila.add(rs.getString("nombre"));
 fila.add(rs.getString("apellidos"));
 fila.add(rs.getString("grupo"));
 fila.add("\n\n\n\n\n\n\n");
 datosTabla.add(fila);
}
```

Una vez que tengamos los datos de la tabla podemos crear un objeto de la clase **DefaultTableModel** de nombre **dtmTabla** que almacene los datos de la JTable.

```
// creo el DefaultTableModel de la JTable
DefaultTableModel dtmTabla = new DefaultTableModel(datosTabla, columnas);
```

Una vez que ya tenemos el modelo de la tabla creado podemos crear la JTable. Nosotros vamos a crear una **JTable** de nombre **tabla**.

```
JTable tabla = new JTable(dtmTabla);
```

Podemos crear un componente de tipo **JScrollPane** de nombre **scrollPane** para meter en él la tabla y que aparezcan barras de desplazamiento si los datos no entran en el espacio destinado a la tabla.

```
// creo un scroll pane y le añado la tabla
JScrollPane scrollPane = new JScrollPane(tabla);
```

Una vez terminado el diseño de la tabla la añadimos al panel principal de la aplicación.

```
// añado el scroll pane al panel principal
contenedor.add(scrollPane, BorderLayout.CENTER);
```

Podemos **ordenar** el contenido de la tabla de varias formas. Para permitir que los datos de una tabla se puedan ordenar al hacer clic sobre la cabecera de una columna escribimos

```
tabla.setAutoCreateRowSorter(true);
```

Si lo que queremos es **ordenar por los datos de una determinada columna** mediante código escribimos

```
// ordeno los datos de la tabla
TableRowSorter<DefaultTableModel> metodoOrdenacion = new TableRowSorter<>(dtmTabla);
tabla.setRowSorter(metodoOrdenacion);
List<RowSorter.SortKey> sortKeys = new ArrayList<>();

// para que ordene por la primera columna (dni en este caso) en Ascendente
int columnIndexToSort = 0;
sortKeys.add(new RowSorter.SortKey(columnIndexToSort, SortOrder.ASCENDING));
metodoOrdenacion.setSortKeys(sortKeys);
metodoOrdenacion.sort();
```

Es importante resaltar que **al reordenar** los datos de la tabla **cambia la posición** en qué se encuentran los mismos.

Otra de las operaciones que se suelen hacer con los registros de una tabla es **filtrar**los para que sólo aparezcan los que cumplen una determinada condición.

Por ejemplo, para que solo aparezcan en una tabla de calificaciones las calificaciones del alumno que esté seleccionado en ese momento escribimos

```
// Defino el método de ordenación
private TableRowSorter<TableModel> metodoOrdenacion;
…
// Después de crear la tabla
// Instanciamos el TableRowSorter y lo añadimos al JTable
metodoOrdenacion = new TableRowSorter<TableModel>(dtmTabla);
tabla.setRowSorter(metodoOrdenacion);
…
// A partir de aquí podemos definir un filtro basado en metodoOrdenacion
// cambio el filtro de la tabla de calificaciones
// muestro solo los datos del alumno con DNI 11111111A
metodoOrdenacion.setRowFilter(RowFilter.regexFilter("11111111A", 0));
// podemos poner un filtro por grupo para sacar solo los alumnos de 1DW3
metodoOrdenacion.setRowFilter(RowFilter.regexFilter("1DW3", 3));
```

En **RowFilter.regexFilter** el primer parámetro indica el valor del filtro (en nuestro caso el DNI del alumno actual), y el segundo la columna de la tabla con la que se compara el valor del filtro. Por ejemplo, si el DNI vale "11111111A" y está en la primera columna de la tabla, el primer caso del ejemplo anterior aplicaría un filtro para que sólo salieran las calificaciones del alumno que tiene ese DNI y el segundo ejemplo mostraría solo los registros de los alumnos del grupo 1DW3.

Para forzar que el contenido del valor por el que queremos filtrar tenga que **coincidir completamente** con el valor del campo debemos usar caracteres de filtrado "^" y "$" junto con el valor. Por ejemplo, si tenemos una variable de tipo String con el valor del dni la expresión de filtrado debería ser

```
metodoOrdenacion.setRowFilter(RowFilter.regexFilter("^"+dni+"$", 0));
```

**Seleccionar una fila de una tabla**. Si queremos hacer algo cada vez que se seleccione una fila de la tabla debemos de usar un **MouseListener** que se encargue de controlar el evento **mouseClicked**. Dentro del evento podemos usar el método **getSelectedRow** para obtener el valor de la fila seleccionada (o -1 si no hay una fila seleccionada). Si hay una fila seleccionada podemos usar el método **getValueAt** para obtener el valor de la columna que queramos.

Por ejemplo, si queremos que al seleccionar un registro de la tabla se actualice el campo de texto txtDNI con el valor de la columna dni del registro que está seleccionado en la tabla escribimos

```
public class EjemploMouseListener extends JFrame implements MouseListener {
…
 tabla = new JTable(dtmTabla);
 tabla.addMouseListener(this);
…
@Override
public void mouseClicked(MouseEvent me) {
 // al seleccionar una fila
 int fila;
 String dni;
 fila = tabla.getSelectedRow(); // o tambien fila = tabla.rowAtPoint(me.getPoint());
 dni = (String) tabla.getValueAt(fila,0);
 txtDNI.setText(dni);
}
```

Cuando ponemos un filtro a una tabla perdemos la relación entre la posición del registro en la tabla y la posición del registros en el DefaultTableModel de la tabla. Para **obtener la posición global de un registro** de la tabla dentro del DefaultTableModel tenemos que usar **convertRowIndexToModel**, que devuelve un valor entero de 0 a n con el valor de la posición global. Por ejemplo, si tenemos una fila seleccionada en una tabla filtrada y queremos obtener su posición global escribimos

```
int filaAbsoluta = tabla.convertRowIndexToModel(fila);
```

Para forzar que sólo se pueda **seleccionar una fila** de una tabla debemos modificar el valor de la propiedad SelectionMode al valor **SINGLE_SELECTION** usando el siguiente código

```
tabla.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
```

**Permitir o impedir la edición de celdas en una JTable**. La edición de celdas en una JTable está activada por defecto. Si queremos hacer que todas las celdas de la tabla no sean editables o que no sea editable alguna columna de la JTable debemos hacerlo por código **modificando el constructor del DefaultTableModel** para sobrescribir el método **isCellEditable**.
Por ejemplo, para hacer que no se pueda editar ninguna celda de la tabla en el constructor del DefaultTableModel escribimos

```
dtmTabla = new DefaultTableModel(datosTabla, columnas){
 @Override
 public boolean isCellEditable(int row, int column) {
  // hago que todas las celdas de la tabla NO sean editables
  return false;
 }
};
```

Si lo que queremos hacer es que sólo se puedan editar las celdas de la tercera columna (columna 2) de la tabla en el constructor del DefaultTableModel escribimos

```
dtmTabla = new DefaultTableModel(datosTabla, columnas){
 @Override
 public boolean isCellEditable(int row, int column) {
  // hago que solo sea editable la tercera columna (columna 2)
  return column == 2;
 }
};
```

Si quiero que sean editables varias columnas, por ejemplo la segunda y la tercera, debo concatenar el valor devuelto por return de la siguiente manera

```
public boolean isCellEditable(int row, int column) {
  // hago que sean editable la segunda  (columna 1) y la tercera columna (columna 2)
  return (column == 1 || column == 2);
 }
};
```

Para **cambiar el valor de una columna de una fila** de una tabla tenemos que usar **setValueAt**. Por ejemplo, si tenemos una fila seleccionada y queremos cambiar el valor de la columna correspondiente a la nota de esa fila escribimos

```
dtmTabla.setValueAt(nota, fila, 2);
```

**Ocultar una columna en una JTable**. En algunas ocasiones puede ser necesario ocultar la información de una o varias columnas de una tabla. Para ocultar una columna ponemos la anchura máxima y mínima de su cabecera y de sus filas de datos a 0. Por ejemplo, para ocultar la primera columna de una tabla (dni) escribimos

```
tabla.getColumnModel().getColumn(0).setMaxWidth(0);
tabla.getColumnModel().getColumn(0).setMinWidth(0);
tabla.getTableHeader().getColumnModel().getColumn(0).setMaxWidth(0);
tabla.getTableHeader().getColumnModel().getColumn(0).setMinWidth(0);
```

**Recorrer una JTable filtrada.** Si tenemos que recorrer una JTable filtrada, por ejemplo, para obtener la media de las calificaciones de un alumno, debemos recorrer la JTable por qué sólo contiene los registros filtrados y no el defaultTableModel qué contiene todos los registros. Por ejemplo, obtener la media de las calificaciones de un alumno escribimos

```
double media = 0.0;
double nota;
int filas = this.tabla.getRowCount();
for(int fila=0;fila<filas;fila++) {
 nota = Double.parseDouble((String) this.tabla.getValueAt(fila, 2));
 media = media + nota;
}
// calculo la media
media = media / filas;
// actualizo la media
this.lblMediaValor.setText(""+media);
```

## Ejercicios Componentes Gráficos para Trabajar con Bases de Datos

8. Crea la clase Java **VentanaJTable** que deriva de JFrame y que contiene un panel de nombre contenedor en el que hay una etiqueta de nombre lblTexto con el valor "Datos de los Alumnos" en la parte superior, una JTable de nombre tabla con los datos de la tabla **alumnos** de la base de datos MySQL **bdalumnos** y una cabecera con los valores "DNI", "Nombre","Apellidos","Grupo", en el centro, y un botón de nombre btnSalir en la parte inferior. Al pulsar el botón Salir se saldrá de la aplicación.

9. Crea la clase Java **VentanaJTableMetadata** que modifica VentanaJTable para que la cabecera de la tabla muestre los valores que tienen las columnas en la base de datos. Para el proceso usa ResultSet.

10. Crea la clase Java **VentanaJTableCalificaciones** que modifica VentanaJTable para que en la tabla aparezcan sólo los datos de las calificaciones del alumno con dni '11111111A'.

11. Crea la clase Java **VentanaJTableInsertar** que modifica VentanaJTable. Añade un JButton de nombre btnInsertar con el texto Insertar en la parte inferior. Al hacer clic sobre btnInsertar inserta un nuevo registro base de datos con los datos dni '00000000A', nombre 'N0', apellidos 'A0', y grupo '1DW3'. Los datos de se deben de actualizar también en la tabla para reflejar los cambios.

12. Crea la clase Java **VentanaJTableBorrar** que modifica VentanaJTableInsertar. Añade un JButton de nombre btnBorrar con el texto Borrar en la parte inferior. Al hacer clic sobre btnBorrar borra el registro

seleccionado en la tabla, si es que hay algún registro seleccionado, de la base de datos. Los datos de la tabla se deben de actualizar para reflejar los cambios.

13. Crea la clase Java **VentanaJTableActualizar** que modifica VentanaJTableBorrar. Añade un JButton de nombre btnActualizar con el texto Actualizar en la parte inferior. Al hacer clic sobre btnActualizar modifica el valor del campo grupo del registro seleccionado en la tabla con el valor 'ADW3', si es que hay algún registro seleccionado, en la base de datos. Los datos de la tabla se deben de actualizar para reflejar los cambios. La columna de la tabla con el campo dni no debe de poderse editar.

## Proyecto Maven con MySQL en Eclipse

Vamos a crear un proyecto Maven en Eclipse de nombre **MavenVentanaJTable** con Group Id **org.apache.maven.archetypes** y como Artifact Id **maven-archetypes-quickstart**. que hace lo mismo que VentanaJTable pero ahora usando la dependencia Maven de MySQL.
Podemos copiar, con cuidado, el contenido de la clase VentanaJTable dentro de la clase **App** o podemos crear toda la clase App desde cero para que sea un JFrame.

Podemos consultar las dependencias Maven de MySQL Connector en https://mvnrepository.com/artifact/mysql/mysql-connector-java

En nuestro caso tenemos que modificar el archivo pom.xml de nuestro proyecto Maven de la siguiente manera

```
<dependency>
 <groupId>mysql</groupId>
 <artifactId>mysql-connector-java</artifactId>
 <version>8.0.28</version>
</dependency>
```

Si queremos crear un fichero **JAR ejecutable que incluya las dependencias** necesarias para poder conectarse a una base de datos MySQL debemos incluir, o modificar si ya existe, en el pom.xml de nuestro proyecto Maven el plugin siguiente

```
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-assembly-plugin</artifactId>
<executions>
	<execution>
		<phase>package</phase>
		<goals>
			<goal>single</goal>
		</goals>
		<configuration>
			<archive>
			<manifest>
		<mainClass>com.fptxurdinaga.MavenVentanaJTable.App</mainClass>
			</manifest>
			</archive>
			<descriptorRefs>
				<descriptorRef>jar-with-dependencies</descriptorRef>
			</descriptorRefs>
		</configuration>
	</execution>
</executions>
</plugin>
```

El plugin maven-jar-plugin lo podemos quitar ya que ahora estamos usando el maven-assembly-plugin para generar el jar.
Ejecutamos el proyecto Maven con un clean install en Run Configurations (marcando opcionalmente Skip Tests) y, si todo ha ido bien, en la carpeta target del proyecto aparecerá el archivo jar **MavenVentanaJTable-0.0.1-SNAPSHOT-jar-with-dependencies.jar** que podemos ejecutar directamente.

## Ejercicios Avanzados Componentes Gráficos para Trabajar con Bases de Datos

14. Crea la clase Java **BDAlumnosCampos** que tiene un campo de texto por cada campo de la tabla Alumnos con su correspondiente etiqueta. Al inicio se cargan los datos de la base de datos en un arrayList y al salir, sólo si los datos han sido modificados, se guardan todos los datos en la base de datos. La carga de los datos del registro actual en los campos de datos se hará usando el método de nombre **actualizarCampos**. La aplicación consta de los siguientes componentes

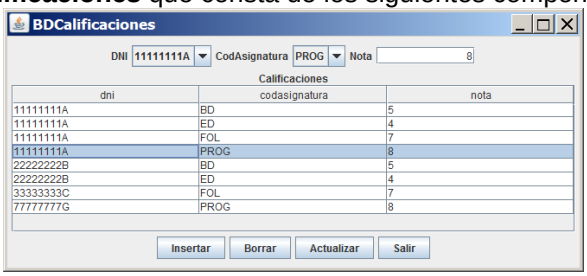

**btnPrimero**. JButton que al hacer clic sobre él permite ir al primer registro de datos (si es que hay).
**btnAnterior**. JButton que al hacer clic sobre él permite ir al registro anterior (si es que hay).
**lblRegistros**. JLabel que muestra el texto inicial "No hay registros" y que si hay registros muestra el texto "Registro: " + posicionActual + " de "+numeroRegistros. Se tiene que actualizar cada vez que cambia la información.
**btnSiguiente**. JButton que al hacer clic sobre él permite ir al registro siguiente (si es que hay).
**btnUltimo**. JButton que al hacer clic sobre él permite ir al último registro (si es que hay).
**btnInsertar**. JButton que al hacer clic sobre él añade un nuevo registro a la base de datos y a la tabla con los datos de los campos de texto. Debe controlar todos los errores que se produzcan.
**btnBorrar**. JButton que al hacer clic sobre él borra el registro actual de la base de datos. Debe controlar todos los errores que se produzcan.
**btnActualizar**. JButton que al hacer clic sobre él modifica los datos del registro actual (menos el dni que es la clave primaria) en la base de datos.
**btnSalir**. JButton que al hacer clic sobre él guarda los datos en la base de datos, sólo si han sido modificados.

15. Crea la clase Java **BDAlumnosCamposResultSet** que modifica la clase BDAlumnosCampos para que las operaciones de inserción, borrado y actualización en la base de datos se hagan en el momento en que se pulsan los botones.

16. Crea la clase Java **BDAlumnosCalificaciones** que modifica BDAlumnosCamposResultSet. Añade una tabla con las calificaciones del alumno que esté seleccionado en ese momento.


17. Crea la clase Java **BDAsignaturasCampos** que modifica BDAlumnosCamposResultSet para que ahora trabaje con los datos de la tabla Asignaturas.

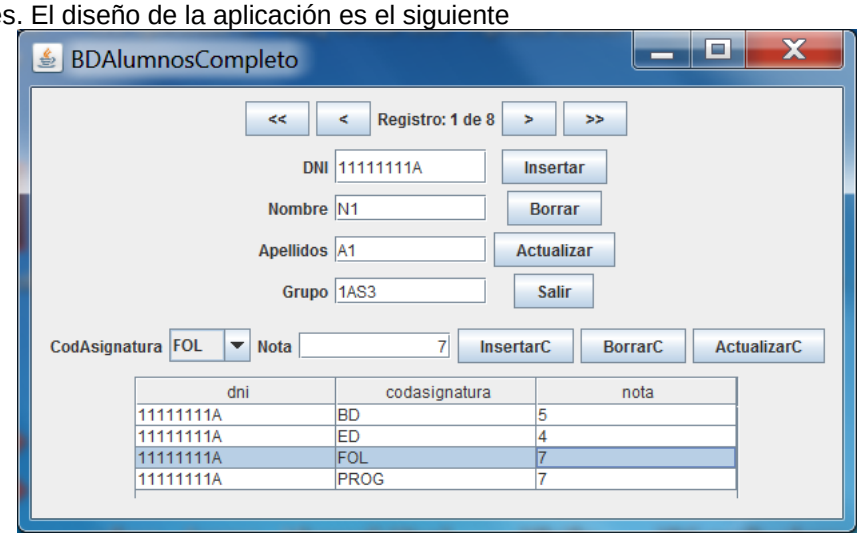
18. Crea la clase Java **BDCalificaciones** que consta de los siguientes componentes




**cmbDni**. JComboBox en la que al inicio se cargan los dnis de todos los alumnos.

**cmbCodAsignatura**. JComboBox en la que al inicio se cargan los codasignatura de todas las asignaturas.
**txtNota**. JTextField en el que aparecerá el valor de la nota. Al inicio tendrá el valor "5".
**btnInsertar**. JButton que al hacer clic sobre él añade un nuevo registro a la base de datos y a la tabla con los datos de los campos de texto. Debe controlar todos los errores que se produzcan.
**btnBorrar**. JButton que al hacer clic sobre él borra el registro seleccionado en la tabla de la tabla y de la base de datos. Debe controlar todos los errores que se produzcan.
**btnActualizar**. JButton que al hacer clic sobre él modifica el valor de la nota del registro actual seleccionado en la tabla con el valor de la nota que hay en txtNota en la tabla de la tabla y en la base de datos.
**btnSalir**. JButton que al hacer sale de la aplicación.
**tabla**. JTable que al hacer clic sobre ella hace que se carguen los datos del registro sobre el que se ha hecho clic y ha quedado seleccionado en cmbDni, cmbCodAsignatura, y txtNota. La tabla no es editable.

19. Crea la clase Java **BDAlumnosCompleto** que es una combinación de BDAlumnosCalificaciones con BDCalificaciones. El diseño de la aplicación es el siguiente




En la tabla aparecerán ahora sólo las calificaciones del alumno que esté seleccionado. La tabla no es editable. Al seleccionar una fila de la tabla se pondrá como elemento seleccionado de cmbCodAsignatura el codasignatura de esa fila y en txtNota la nota de esa fila. Los datos se actualizarán en la base de datos al pulsar los botones de Insertar, Borrar o Actualizar.