MEDAC
Instituto Oficial de Formación Profesional

PROGRAMACIÓN

Bases de datos relacionales

17

ÍNDICE

OBJETIVOS

Conocer los diferentes modelos de bases de datos.

Instalar un gestor de bases de datos relacionales.

Desarrollar programas con acceso a bases de datos relacionales.

Insertar datos en una base de datos relacional.

Recuperar información de una base de datos relacional.

Borrar y modificar datos de una base de datos relacional.

/ 1. Introducción y contextualización práctica

En esta unidad vamos a recordar el concepto de base de datos y profundizaremos sobre qué es una base de datos relacional.

Vamos a ver cómo podemos configurar un proyecto de NetBeans para poder tener acceso a una base de datos relacional y qué bibliotecas tenemos que usar para ello.

También vamos a ver cómo se realizan las operaciones básicas de inserción, selección, borrado y actualización de datos mediante consultas SQL.

Por último, vamos a ver qué excepciones deberemos manejar a la hora de acceder a una base de datos relacional.

Escucha el siguiente audio en el que planteamos el caso práctico que iremos resolviendo a lo largo de esta unidad.

Fig. 1. La importancia de la buena gestión del dato.

Audio Intro. "Eficiencia en el acceso a datos"

http://bit.ly/2K2DXK4

/ 2. Introducción a las bases de datos relacionales

Las bases de datos nos proveen de un sistema eficiente para el tratamiento de grandes cantidades de información.

Éstas nacieron para mejorar todas las carencias del almacenamiento de información en ficheros. Recordemos de unidades anteriores, que los ficheros, cuando se trata de poco volumen de información, pueden ser muy útiles, pero cuando hablamos de grandes cantidades de datos, añadiendo a esto la necesidad de su acceso secuencial, no dan una respuesta óptima.

Hay muchos tipos de bases de datos, pero en este tema nos vamos a centrar en las **bases de datos relacionales**, que son las que cumplen con el modelo de datos relacional.

Dentro de las bases de datos relacionales podemos distinguir las siguientes **características comunes**:

- Los datos de almacenan en tablas, las cuales están relacionadas con otras tablas.
- Los nombres de las tablas no pueden repetirse.
- Cada tabla se compone de un conjunto de campos.
- Cada tabla debe tener obligatoriamente una clase primaria, es decir, un registro mediante el cual poder distinguir el resto de la fila.
- Las tablas deben cumplir una serie de reglas de integridad.

Las operaciones que vamos a poder realizar en una base de datos relacional son las siguientes:

- Inserción de datos.
- Consulta de datos.
- Borrado de datos.
- Modificación de datos.

Con esas operaciones básicas podremos gestionar la información de cualquier aplicación que necesite el almacenamiento de datos en una base de datos.

Fig. 2. Bases de datos.

/ 3. Instalación de un gestor de bases de datos relacionales

Para poder trabajar con bases de datos relacionales, lo primero que deberemos hacer será instalar un **sistema gestor de bases de datos relacionales.**

En nuestro caso vamos a elegir ***XAMPP*** por su simplicidad de uso y gran potencia, ya que vamos a tener un sistema gestor de bases de datos relaciones que utiliza MySQL sin tener que instalar ningún programa pesado.

***XAMPP* es un programa de *software* libre que contiene un sistema gestor de bases relacionales con MySQL, un servidor Apache, e intérpretes para los lenguajes Perl y PHP.** Lo podemos descargar de forma gratuita de su página web:

https://www.apachefriends.org/es/download.html

Una vez descargado, su instalación es muy sencilla, solo tendremos que pulsar el botón de 'siguiente' y se instalará automáticamente, sin ninguna configuración especial.

Una vez lo tengamos instalado tendremos una interfaz como se muestra en la siguiente figura.

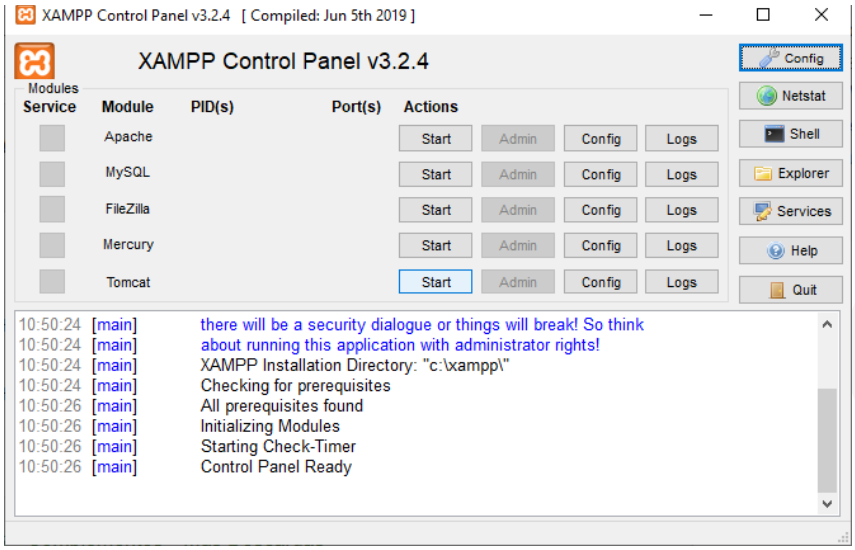

Fig. 3. XAMPP.

Para ejecutar MySQL basta con pulsar en el botón ***Start*** de las secciones MySQL y Apache, y ya tendremos nuestro sistema gestor de bases de datos operativo.

Para gestionar las bases de datos MySQL pulsaremos sobre el botón ***Admin*** de MySQL (que hasta que no llevemos a cabo el paso anterior, aparecerá inactivo), y acto seguido observaremos que se habilitará *PHPmyadmin* para poder trabajar.

Audio 1. "¿Qué gestor de bases de datos elegir?"
http://bit.ly/3nzqIOS

/ 4. Caso práctico 1: "¿Qué tipo de base de datos usar?"

Planteamiento: Pilar y José están repasando lo que han aprendido en la asignatura de Bases de Datos y están recordando que existen muchos tipos de bases de datos diferentes.

En esta unidad que estamos estudiando actualmente, al ser una unidad "cruce" de las asignaturas de Programación y Bases de Datos, pueden utilizar lo aprendido en ambas para beneficiarse, pero las dudas no tardan en llegar. "Pilar, no recordaba que hubiese tantos tipos de bases de datos", comenta José. Pilar también está un poco desconcertada, "y si tuviésemos que realizar una aplicación real, ¿cuál de todas esas deberíamos usar?"

Fig. 4. Información almacenada.

Nudo: ¿Qué piensas al respecto? Con la gran cantidad de tipos de bases de datos que existen, ¿cuál será mejor a la hora de usarla en un programa?

Desenlace: Esta pregunta siempre va a estar presente a la hora de realizar de un proyecto real. En todo proyecto que se precie habrá de por medio una base de datos, ya que con toda probabilidad el almacenamiento de información, y no precisamente poca, será más que necesario. Sencillamente, imprescindible.

Normalmente, puede que para la gran mayoría de los proyectos de desarrollo en *backend*, lo que se use sea una base de datos relacional, ya que cubren las necesidades habituales que van a demandar la gran mayoría de programas del mercado.

No obstante, si existen otros tipos de bases de datos es por alguna razón. Por ejemplo, las bases de datos no relacionales basadas en XML o en JSON se usan mucho en sistemas web, donde una base de datos relacional no puede presentar la solución óptima.

Por tanto, no podemos afirmar que haya un tipo de base de datos mejor que otra de forma general.

Habrá que analizar en cada caso qué tipo de desarrollo se va a llevar a cabo, con qué necesidades, y en función de esto, elegir la base de datos que mejor se adapte.

/ 5. Conectores Java-MySQL

Los conectores de Java con MySQL son una **biblioteca que nos va a proporcionar las herramientas y métodos necesarios para poder conectar un programa escrito en Java con una base de datos MySQL.**

Podemos descargar los conectores de la página web oficial de MySQL:

https://dev.mysql.com/downloads/connector/j/

Para proceder a su descarga:

1. Primeramente, debemos seleccionar en el combo donde pregunta por el sistema operativo, ***Platform Independent***.

2. A continuación, descargamos el fichero comprimido que se habilita tras realizar la selección anterior. Éste, contendrá un **fichero *Jar***, que será la biblioteca que necesitamos agregar a nuestros proyectos para conectar con MySQL.

3. Una vez que tengamos **descargado el conector** de Java con MySQL, **lo guardamos** en una carpeta dentro de nuestro proyecto.

4. A continuación, tendremos que importarlo como biblioteca en cada uno de los proyectos que lo necesiten. Para ello, pulsamos con el botón derecho en la zona izquierda de NetBeans, en el paquete denominado ***Libraries*** **o Bibliotecas, y seleccionamos *Agregar archivo JAR*.**

5. Acto seguido, buscamos en el proyecto donde tengamos descargado el conector, lo seleccionamos, y listo.

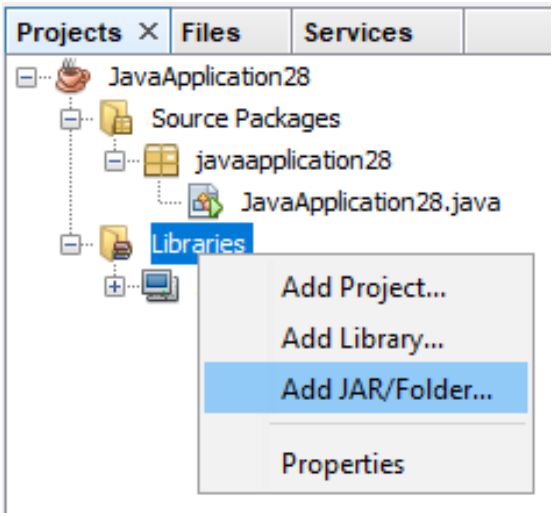

Fig. 5. Añadir librería.

6. El conector importado **nos provee de las siguientes clases** para la manipulación de información en una base de datos MySQL:

CLASE	FUNCIÓN
Connection	Clase que nos permitirá instanciar objetos donde se almacenará nuestra conexión.
Statement	Clase que nos permitirá instanciar objetos que utilizaremos para realizar las consultas SQL.
Result	Clase que nos permitirá instanciar objetos que nos devolverán los resultados de las consultas SQL.
SQLException	Excepción propia de SQL.

Tabla 1. Clases para acceso a base de datos MySQL.

/ 6. Conexión y desconexión con la base de datos

Cuando trabajamos con bases de datos, sean cual sean éstas, vamos a tener **dos operaciones que son esenciales**, la **conexión** y la **desconexión** de la misma.

El primer paso a realizar siempre deberá ser conectarse a la base de datos deseada, para así poder llevar a cabo la acción requerida, y a continuación, desconectarnos de la misma.

En resumen, **siempre que vayamos a realizar una operación sobre una base de datos deberemos**:

- Conectar a la base de datos.
- Realizar la operación pertinente.
- Desconectar de la base de datos.

Si bien el esquema anterior no es obligatorio, es muy recomendable seguirlo, ya que el no cerrar conexiones a las bases de datos puede provocar inconsistencias en las mismas.

Para conectar a la base de datos deseada lo haremos de la siguiente forma:

```
String BD = "personas";
String USUARIO = "root";
String PASS = "";
String HOST = "localhost";

Calendar now = Calendar.getInstance();
TimeZone zonahoraria = now.getTimeZone();
Connection connection = (Connection) DriverManager.getConnection(
        "jdbc:mysql://" + HOST + "/" + BD + "?user="
        + USUARIO + "&password=" + PASS
        + "&useLegacyDatetimeCode=false&serverTimezone="
        + zonahoraria.getID());
```

Fig. 6. Conectando a una base de datos.

Aquí **conectamos** mediante el **método *getConnection*** a la base de datos *personas*, con el usuario *root*, con contraseña vacía.

Observamos en la figura que también se lleva a cabo un uso de la zona horaria. Esto es una cuestión de configuración nada importante. Para desconectar de la base de datos simplemente llamaremos al **método *close* del objeto de la clase *Conection*.**

/ 7. Inserción de datos

En **SQL para insertar datos en una tabla se utiliza el comando *INSERT***, al cual le pasaremos el nombre de la tabla, los nombres de las columnas y los valores a insertar. Por ejemplo:

```
INSERT INTO persona (DNI, nombre, apellidos) VALUES ('147A', 'Pepe', 'López')
```

Código 1. Inserción de datos.

En **Java, vamos a usar la propia sentencia *INSERT* solo que tendremos que almacenarla en un objeto del tipo *String*.**

En este punto hay que tener un **extremo cuidado** con el uso de las **comillas simples** que preceden a los valores del tipo cadena. Hay que recordar que cuando no se trate de una cadena, sino de un número tanto entero, como real, éstos conllevarán dichas comillas simples.

Para ejecutar las sentencias *INSERT* vamos a seguir el siguiente código:

```
Statement stmt = connection.createStatement();
int filas_afectadas = stmt.executeUpdate(consulta_insert);
```

Código 2. Inserción de datos usando comillas simples.

1. El primer paso es crear un **objeto de la clase *Statement***, el cual va a obtener la base de datos para poder realizar una acción sobre ella.
2. A continuación, una vez hecho esto, sobre el objeto creado utilizaremos el **método *executeUpdate***, al cual le pasaremos un String con la consulta INSERT que deseamos ejecutar. Este método devolverá un entero, que representará el número de filas afectadas por la sentencia ejecutada. En este caso siempre va a ser una, ya que insertamos una fila nueva.
3. Este método **lanzará una excepción *SQLException***, por lo que tendremos que tratarla con un bloque *try-catch*.

Fig. 7. Gestionando una base de datos.

/ 8. Consulta de datos

En SQL **para consultar datos de una tabla se utiliza el comando *SELECT*,** al cual le pasaremos las columnas que deseamos consultar, y una serie de restricciones a las mismas mediante las instrucciones *WHERE*, *GROUP BY, ORDER BY*, etc., por ejemplo:

```
SELECT * FROM persona WHERE edad >= 18
```

Código 3. Consultar datos de una tabla.

Igualmente, que con ocurría con *INSERT,* utilizaremos la propia **sentencia *SELECT* y la almacenaremos en un objeto del tipo *String***.

Las sentencias *SELECT* que podremos lanzar a las bases de datos podrían complicarse tanto como necesitemos. Esto dependerá de las restricciones que impongamos a la consulta. De igual forma que con las demás sentencias, hay que tener un **extremo cuidado** con el uso de las **comillas simples** que preceden a los valores del tipo cadena, y recordar que cuando no se trate de una cadena, sino de un número tanto entero como real, se tendrá que hacer uso de dichas comillas simples.

Para ejecutar las sentencias *SELECT* vamos a seguir el siguiente código:

```
Statement stmt = connection.createStatement();
ResultSet rset = stmt.executeQuery(consulta_select);
```

Código 4. Ejecutar las sentencias SELECT.

1. El primer paso sigue siendo la creación de un **objeto de la clase *Statement*** para poder lanzar operaciones sobre la base de datos.

2. A continuación, sobre el objeto creado utilizaremos el **método *executeQuery*,** al cual le pasaremos un *String* con la consulta *SELECT* que deseamos ejecutar. Este método devolverá un objeto de tipo ***ResultSet*,** que contendrá el resultado de la ejecución de la consulta.

3. Al igual que en el apartado anterior, este método **lanzará una excepción *SQLException*,** por lo que tendremos que tratarla con un bloque *try-catch*.

4. Para recorrer y obtener los valores de un **objeto de tipo *ResultSet*** utilizaremos el siguiente código, en donde observamos que tendremos **un método *get* para cada uno de los diferentes datos**:

```
ResultSet resultado = stmt.executeQuery(consulta_select);
while(resultado.next())
{
        String nombre = resultado.getString("nombre");
        int edad = resultado.getInt("edad");
}
```

Código 5. Inserción de datos usando comillas simples.

Vídeo 1. "Ejecución de una consulta SELECT"
https://bit.ly/2XB27yl

/ 9. Borrado de datos

En SQL, **para borrar datos de una tabla se utiliza el comando *DELETE***, al cual le pasaremos el nombre la tabla de la que queremos eliminar los datos y una serie de restricciones con la instrucción *WHERE*, por ejemplo:

```
DELETE FROM persona WHERE edad >= 18
```

Código 6. Borrar datos de una tabla.

Para su uso en Java, emplearemos esta misma sentencia *DELETE,* pero tendremos que **almacenarla en un objeto del tipo *String*.**

Es vital que las sentencias *DELETE* conlleven aparejado 'su' *WHERE* correspondiente, ya que, si no lo contemplaran, **borrarían todos los datos** de la tabla que se les hubiera indicado.

Nuevamente, al igual que ocurre con las demás sentencias, hay que tener un **extremo cuidado** con el uso de las **comillas simples** que preceden a los valores del tipo cadena, y recordar que cuando no se trate de una cadena, sino de un número tanto entero como real, éstos conllevarían comillas simples.

Para ejecutar las sentencias *DELETE* vamos a seguir el siguiente código:

```
Statement stmt = connection.createStatement();
int filas_afectadas = stmt.executeUpdate(consulta_delete);
```

Código 7. Borrar datos de una tabla.

1- El primer paso es **crear un objeto de la clase *Statement***, el cual va a obtener la base de datos para poder realizar una acción sobre ella.

2- Acto seguido, sobre el objeto creado utilizaremos el **método *executeUpdate***, al cual le pasaremos un *String* con la consulta *DELETE* que deseamos ejecutar. Este método devolverá un entero, que representará el número de filas afectadas por la sentencia ejecutada, es decir, la cantidad de filas que han sido eliminadas con la consulta.

3- Este método **lanzará una excepción *SQLException***, por lo que tendremos que tratarla con un bloque *try-catch*.

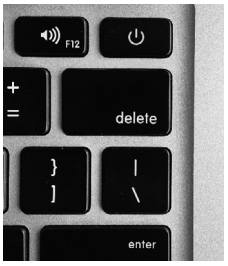

Fig. 8. El borrado es una acción muy peligrosa.

/ 10. Modificación de datos

En SQL, **para modificar datos de una tabla se utiliza el comando *UPDATE*,** al que le pasaremos el nombre de la tabla, los datos a modificar y una serie de restricciones con la instrucción *WHERE*, por ejemplo:

```
UPDATE persona SET edad = 20 WHERE nombre LIKE F%
```

Código 8. Modifica datos de una tabla.

Al igual que ocurría con DELETE, **almacenaremos el resultado de la sentencia en un objeto de tipo *String*.**

Es vital que conlleven aparejado 'su' *WHERE* correspondiente, ya que, si no lo contemplaran, **actualizarían todos los datos de la tabla** que se les hubiera indicado. En el caso de uso de enteros y reales, éstos, con **comillas simples**.

Para ejecutar las sentencias *UPDATE* vamos a seguir el siguiente código:

```
Statement stmt = connection.createStatement();
int filas_afectadas = stmt.executeUpdate(consulta_delete);
```

Código 9. Modifica datos de una tabla.

1. El primer paso es **crear un objeto de la clase *Statement***, el cual va a obtener la base de datos para poder realizar una acción sobre ella.

2. Una vez hecho esto, sobre el objeto creado utilizaremos el **método *executeUpdate***, al cual le pasaremos un *String* con la consulta *UPDATE* que deseamos ejecutar. Este método devolverá un entero, que representará el número de filas afectadas por la sentencia ejecutada, es decir, la cantidad de filas que han sido actualizadas con la consulta.

3. Este método, ya sabemos que **lanzará una excepción *SQLException***, por lo que tendremos que tratarla con un bloque *try-catch*.

Fig. 9. No olvides el WHERE en los UPDATE.

/ 11. Aplicando la POO

Ya hemos visto cómo podemos ejecutar las instrucciones básicas para el tratamiento de la información en una base de datos relacional, siendo éstas: *INSERT, DELETE, UPDATE y SELECT*.

En el caso del comando ***SELECT* tiene su propio método** para ejecutarlo, pero los comandos ***INSERT, DELETE y UPDATE* se ejecutan con el mismo método.**

Siguiendo la filosofía de la programación orientada a objetos, podremos crear una única clase para ejecutar dichos comandos, es decir, que contenga un método para ejecutar los comandos *SELECT* y un método genérico para poder ejecutar los *INSERT, DELETE y UPDATE*.

Así, podemos crear la **clase *ConexionMySQL*** que constará de los siguientes métodos:

- **Un constructor** al que le pasemos la base de datos a la que nos queremos conectar, nuestro usuario, y contraseña.
- **Un método conectar**, que conectará con la base de datos. Este método lanzará una excepción ***SQLException*.**
- Un **método desconectar,** que desconectará de la base de datos. Este método lanzará una excepción ***SQLException*.**
- Un **método *ejecutarSelect***, que ejecutará una consulta *SELECT*. A este método le pasaremos un *String* con la consulta a ejecutar y devolverá un ***ResultSet*** con el resultado de la ejecución. Este método lanzará una excepción *SQLException*.
- Un método ***ejecutarInsertDeleteUpdate*,** que ejecutará tanto consultas *INSERT, DELETE* como *UPDATE*. A este método le pasaremos un ***String*** con la consulta a ejecutar y devolverá un entero con el número de filas afectadas. Este método lanzará una excepción *SQLException*.

Puedes consultar el código completo de esta clase en el apartado de Recursos del tema.

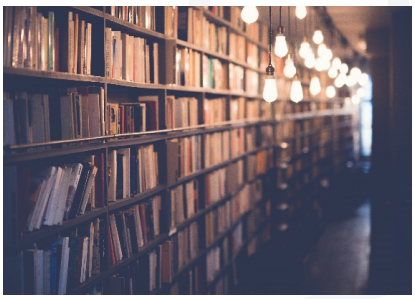
Fig. 10. Las bibliotecas pueden ser consideradas bases de datos.

/ 12. Caso práctico 2: "Optimizando los accesos a la base de datos"

Planteamiento: Una vez que Pilar y José han empezado a realizar el proyecto de la clínica veterinaria, el trabajo se ha ido desarrollando más rápido de lo que esperaban. José, bastante contento, le comenta a Pilar que ya tiene bastante avanzado del proyecto y Pilar le pide que le deje ver su código porque ella está teniendo algún problema a la hora de tratar con la base de datos.

Al ver el código de José, Pilar se extraña, "has repetido muchas veces el código para la ejecución de las consultas a la base de datos". José, sorprendido, le pregunta qué hay de malo en ello.

Nudo: ¿Qué piensas al respecto? ¿Crees que José está realizando correctamente el problema repitiendo código? ¿O es Pilar quien está equivocada?

Desenlace: Ya hemos visto desde las primeras unidades de la asignatura que una de las grandes ventajas de la programación dirigida a objetos es la reutilización de código. A la hora de conectar a la base de datos esto no va a ser menos y deberemos repetir el código que escribamos lo mínimo posible, para así poder hacer los programas más estructurados.

En el caso del acceso a las bases de datos, esto se puede realizar de una forma bastante fácil creando una clase para tal efecto. Es decir, podremos crear una clase que nos gestione todo lo que tenga que ver con el acceso a la base de datos. Así, con ella, podremos conectarnos, desconectarnos, realizar consultas, inserciones, etc. Además, si a esto le unimos el Modelo Vista-Controlador, conseguiremos que nuestros programas estén optimizados y sean muy fáciles de entender.

Fig. 11. Hay que evitar repetir código.

/ 13. Gestión de excepciones

Las operaciones con bases de datos son operaciones muy propensas a fallar ya que intervienen muchos elementos en el proceso, y por ello, utilizaremos la **gestión de excepciones prácticamente con todas las operaciones** que hemos visto.

Los motivos por los que podemos tener una excepción en el tratamiento de bases de bases de datos, entre muchos otros, pueden ser:

- No se ha conectado a la base de datos.
- Los datos de conexión son incorrectos, bien porque la base de datos no existe o porque el usuario o la contraseña son incorrectos.
- La instrucción SQL que se quiere lanzar a la base de datos tiene fallos sintácticos.
- La base de datos se ha corrompido.
- No se tienen permisos para realizar alguna de las operaciones.

Como bien sabemos, el tratamiento de excepciones se realizará **mediante un bloque *try-catch*.**

En el caso de las bases de datos, al igual que pasaba con los ficheros, sería conveniente cerrar **las conexiones en el bloque *finally***, para que se cerraran tanto si han ocurrido errores como si no. Un ejemplo de ejecución de una consulta *SELECT* a una base de datos gestionada mediante la clase *ConexionMySQL* y un bloque *try-catch-finally* podría ser el que se muestra en la siguiente figura.

Para que podamos cerrar la conexión dentro del bloque *finally* tendremos que declarar el objeto de la clase *ConexionMySQL* fuera del *try-catch*.

```
onexionMySQL conexion = new ConexionMySQL("root", "", "personas"
:y {
   conexion.conectar();

   String consulta = "SELECT * FROM persona";

   ResultSet resultado = conexion.ejecutarSelect(consulta);
catch (SQLException ex) {
   System.out.println("Error: " + ex.toString());
finally {
   try {
       conexion.desconectar();
   } catch (SQLException ex) {
       System.out.println("Error: " + ex.toString());
   }
```

Fig. 12. Ejemplo de conexión con tratamiento de excepciones.

/ 14. Bases de datos e interfaz gráfica

El trabajar con una interfaz gráfica es algo fundamental hoy en día, ya que no vamos a desarrollar, por ejemplo, un programa de gestión, y lo vamos a entregar en consola, eso sería impensable.

Generalmente **el uso de la interfaz gráfica va unido al de las bases de datos,** ya que conseguiremos que todo sea mucho más visual y fácil de entender para el usuario.

Igual que vimos en la unidad de interfaces gráficas, para las bases de datos, también podremos introducir la información mediante cajas de texto, tendremos botones que nos permitirán conectarnos a la base de datos, desconectarnos, ejecutar una consulta, etc.

Quizás lo más complicado sea el mostrar los datos de una consulta *SELECT*, ya que tendremos que utilizar una tabla, y éstas son un poco más complicadas de utilizar que los demás componentes.

Haciendo un poco de repaso, para escribir en una tabla, a groso modo, necesitábamos obtener su modelo, introducir los datos en un *array* y luego ir agregando esos datos en filas.

Estos pasos son exactamente los mismos para el caso de utilizar una base de datos.

Aquí podremos **utilizar el Modelo Vista-Controlador** para terminar de ordenar a la perfección los programas.

Imaginemos que tenemos una tabla para mostrar los datos de personas y queremos cumplimentar su nombre, apellidos y teléfono. El código para obtener esos datos y rellenar la tabla podría ser el que se muestra en la siguiente figura.

Podemos ver cómo obtenemos el modelo de la tabla y recorremos *un array* llamado *personas* que está compuesto de objetos de tipo *Persona*. Al recorrerlo obtenemos los datos necesarios de las personas y los almacenamos en otro *array* que será la fila, agregándolo al modelo en último lugar.

```
// Introducimos los datos en la tabla
DefaultTableModel modelo = (DefaultTableModel) tablaDatos.getModel();
for(Persona per : personas)
{
    ArrayList<String> fila = new ArrayList<>();
    fila.add(per.getNombre());
    fila.add(per.getApellidos());
    fila.add(per.getTelefono());

    modelo.addRow(fila.toArray());
}
```

Fig. 13. Rellenando una tabla.

Vídeo 2. "Ejemplo completo de acceso con interfaz gráfica"
https://bit.ly/39kPbSW

/ 15. Resumen y resolución del caso práctico de la unidad

A lo largo de esta unidad hemos dado un pequeño repaso a lo que son las **bases de datos** y las ventajas que nos proveen. Hemos visto que cuando conectamos nuestros programas con una **base de datos relacional** el acceso a la información se puede realizar de una forma muy sencilla mediante la biblioteca que nos da acceso a ella.

También hemos aprendido cómo podemos **recuperar información de la base de datos, así como insertarla, borrada y modificarla**, para poder cubrir todo el abanico de necesidades básicas.

Lo siguiente que hemos estudiado, ha sido cómo podemos gestionar de forma eficiente todas las **excepciones** que se pueden producir en el acceso a una base de datos.

Por último, hemos visto cómo podemos usar una **interfaz gráfica** para mostrar la gestión de la información de la base de datos.

Resolución del caso práctico inicial

Cualquier programa desarrollado para un entorno profesional, deberá poder dar un acceso eficiente a la información.

En el caso práctico nuestros amigos deben poder acceder a una información que pudiera llegar a ser muy extensa, ya que... ¿cuántos animales pueden pasar por una clínica veterinaria si ésta mantiene una actividad normal? La respuesta es muchos, por lo que se intuye que se deberá tratar un volumen considerable de información.

Ya estudiamos en unidades pasadas que podríamos utilizar ficheros para el almacenamiento de información, pero estos pueden llegar a ser muy lentos, además de secuenciales. La solución a todos estos problemas está en el uso de una base de datos, ya que éstas están totalmente optimizadas para almacenar grandes volúmenes de información, ofreciendo una forma rápida de acceso a las mismas.

Fig. 14. Información.

/ 16. Bibliografía

Colaboradores de Wikipedia. (2020, mayo 25). Base de datos - Wikipedia, la enciclopedia libre. Recuperado 3 de junio de 2020, de https://es.wikipedia.org/wiki/Base_de_datos