

ACCESO A BASES DE DATOS DESDE JAVA

La empresa **MANEMPSA**, una empresa encargada de proporcionar servicios de mantenimiento a otras empresas.

En este ejercicio se creará la base de datos MANEMPSA formada por las tablas siguientes usando el programa phpMyAdmin..

Tabla Coches

<u>Matrícula</u>	Marca	Modelo	Año	DNI

- Matrícula VARCHAR de 10, clave primaria.
- Marca VARCHAR de 50.
- Modelo VARCHAR de 50.
- Año int, hace referencia al año de compra.
- DNI VARCHAR de 15, clave foránea.

Tabla Trabajadores

<u>DNI</u>	Nombre	Apellidos	Sueldo	Fecha	Matrícula

- DNI VARCHAR de 15, clave primaria.
- Nombre VARCHAR de 50.
- Apellidos VARCHAR de 50.
- Sueldo Double.
- Fecha tipo DATE.
- Matrícula VARCHAR de 10.

Tabla Servicios

<u>Número</u>	Fecha	Tipo	Cantidad	Comentario	DNI	CIF

- Número int, es autoincrement y clave primaria.
- Fecha tipo DATE.
- Tipo VARCHAR de 50.

- Cantidad Double.
- Comentario Text.
- DNI VARCHAR de 15.
- CIF VARCHAR de 15.

Tabla Clientes

CIF	Nombre	Dirección	Tfno 1	Tfno 2

- CIF VARCHAR de 15, clave primaria.
- Nombre VARCHAR de 100.
- Dirección VARCHAR de 100.
- Tfno1 VARCHAR de 15.
- Tfno2 VARCHAR de 15.

Introducir los datos siguientes en las tablas:

Coches : Tabla					
	Matricula	Marca	Modelo	Año	DNI
	3322-ASR	SEAT	Ibiza	2000	21.123.123-A
	4433-ABB	CITROEN	Saxo	2001	12.321.567-B
▶				0	

Trabajadores : Tabla						
	DNI	Nombre	Apellidos	Sueldo	Fecha	Matricula
	21.123.123-A	Ana	Ruiz	1200	02/03/2002	3322-ASR
	12.321.567-B	Juan	Pérez	1120	04/05/2002	4433-ABB
▶				0		

Servicios : Tabla							
	Numero	Fecha	Tipo	Cantidad	Comentario	DNI	CIF
▶	1	12/04/2004	Limpieza	300		21.123.123-A	B11223212
	2	22/05/2005	Fontanería	238	Arreglo tuberías	12.321.567-B	B22334466
	3	21/12/2005	Electricidad	130	Revisión cableado	21.123.123-A	B33221111
	4	10/11/2006	Fontanería	250		12.321.567-B	B11223212
*	(Autonumérico)			0			

Clientes : Tabla					
	CIF	Nombre	Dirección	Tfno1	Tfno2
	B11223212	Seguros Segasa	C/Ancha 2	956344334	629234323
	B22334466	Academia La Plata	C/La Plata 10	956302323	
	B33221111	Papelería Cuatro	C/Larga 8	956305060	
▶					

Lenguaje de Consulta SQL

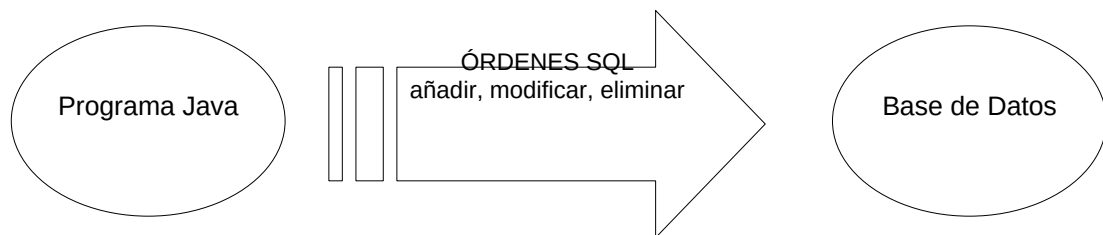
El lenguaje SQL (lenguaje de consulta estructurado) es un lenguaje que permite “actuar” sobre una base de datos.

Con este lenguaje se pueden construir órdenes que permiten hacer lo siguiente (entre otras cosas):

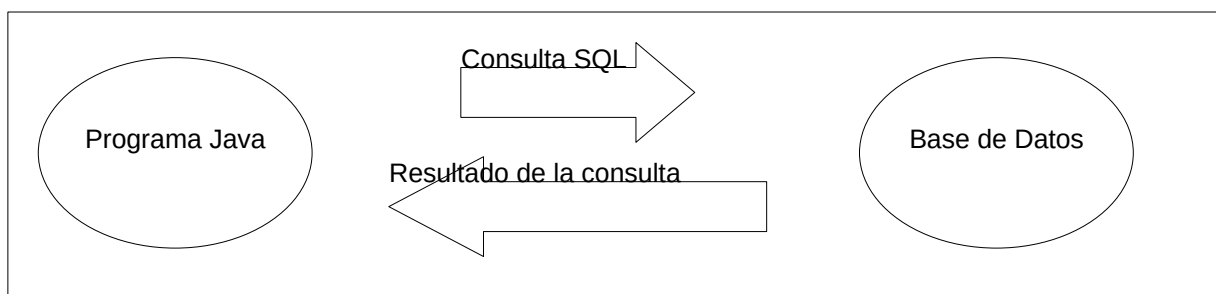
- Añadir registros a las tablas.
- Modificar registros de las tablas.
- Eliminar registros de las tablas.
- Realizar Consultas sobre las tablas.

Gracias a este lenguaje, se construirán órdenes desde nuestra aplicación java, que se aplicarán a la base de datos, actuando sobre ella.

Las órdenes de *añadir*, *modificar*, *eliminar* realizan cambios dentro de la base de datos, pero no devuelven nada al programa java.



Por otro lado, cuando se da una orden de *realizar una consulta*, la base de datos nos devuelve el resultado de dicha consulta:



Gracias a este lenguaje, nuestra aplicación tiene dominio total sobre la base de datos. Puede actuar sobre ella introduciendo nuevos datos, o modificando los que había, o eliminándolos. También puede extraer información de ella accediendo a las consultas de la base de datos o realizando nuevas consultas.

(Para recordar.)

A continuación se comentarán las reglas básicas de este lenguaje.

Creación de consultas en SQL

Se empezará estudiando como realizar consultas sobre una base de datos usando el lenguaje SQL (más adelante se verá como realizar consultas de acción: añadir, modificar eliminar)

Código base en SQL para realizar consultas

Para consultar una base de datos usará un código general como el que sigue:

```
SELECT campos a visualizar
FROM tablas donde se encuentran dichos campos
WHERE condiciones que deben cumplir los registros
ORDER BY forma de ordenar la consulta;
```

Como puede ver, una consulta en SQL tiene cuatro partes (SELECT, FROM, WHERE y ORDER BY) de las cuales solo las dos primeras son obligatorias.

Se debe mantener el orden indicado. Es decir, primero SELECT, luego FROM, luego WHERE y luego ORDER BY.

Este código debe terminar siempre con punto y coma ;

A continuación se verán ejemplos de uso de este código general.

Visualizar una tabla entera (todos los campos y todos los registros)

Ejemplo: "Visualizar la tabla Clientes"

```
SELECT *
FROM clientes;
```

Observa, el * significa ver todos los campos. En el FROM se indica la tabla que se quiere ver. Observa como hay que terminar con un ;

Visualizar algunos campos de una tabla (algunos campos y todos los registros)

Ejemplo: "Visualizar CIF, nombre y Direccion de todos los clientes"

```
SELECT clientes.CIF, clientes.nombre, clientes.direccion  
FROM clientes;
```

Observa como se indican los campos a visualizar en la cláusula SELECT. Se indica la tabla y luego el nombre del campo, separados por un punto.

Visualizar solo aquellos registros de la tabla que cumplan una condición

Ejemplo: “Visualizar todos los campos de aquellos trabajadores que cobren un sueldo superior a los 1000 euros”

```
SELECT *  
FROM trabajadores  
WHERE trabajadores.sueldo > 1000;
```

Observa el uso de la cláusula WHERE para aplicar una condición al resultado.

Ejemplo: “Visualizar el nombre, apellido y sueldo de aquellos trabajadores que cobren un sueldo entre 800 y 2000 euros”

```
SELECT trabajadores.nombre, trabajadores.apellidos,  
trabajadores.sueldo  
FROM trabajadores  
WHERE trabajadores.sueldo BETWEEN 800 AND 2000;
```

Observa el uso de BETWEEN – AND para indicar que el sueldo esté entre 800 y 2000

Nota. Más adelante en este ejercicio guiado se muestran las distintas posibilidades que tenemos a la hora de indicar criterios en la cláusula WHERE

Visualizar el contenido de una tabla ordenado

Ejemplo: “Visualizar la tabla de trabajadores ordenada por sueldo de menor a mayor”

```
SELECT *  
FROM trabajadores  
ORDER BY trabajadores.sueldo ASC;
```

Observa el uso de la cláusula ORDER BY para indicar que se ordene por sueldo. La palabra ASC indica “ascendente” (de menor a mayor)

Ejemplo: “Visualizar nombre, apellidos y sueldo de los trabajadores ordenados por sueldos de mayor a menor”

```
SELECT trabajadores.nombre, trabajadores.apellidos,  
trabajadores.sueldo  
FROM trabajadores  
ORDER BY trabajadores.sueldo DESC;
```

Observa el uso de DESC para indicar una ordenación descendente.

Ejemplo: “Visualizar nombre, apellidos y sueldo de los trabajadores que cobren más de 1000 euros, ordenados por apellidos y nombre”

```
SELECT trabajadores.nombre, trabajadores.apellidos,  
trabajadores.sueldo  
FROM trabajadores  
WHERE trabajadores.sueldo > 1000  
ORDER BY trabajadores.apellidos ASC, trabajadores.nombre ASC;
```

Observa aquí como ordenar por dos campos, primero por apellidos y luego por nombre. Esto significa que aquellos trabajadores que tengan el mismo apellido serán ordenados por nombre.

Visualizar datos de varias tablas

Ejemplo: “Visualizar todos los servicios. Interesa que aparezca el nombre del trabajador que hizo el servicio, la fecha del servicio realizado y el tipo de servicio”

```
SELECT trabajadores.nombre, servicios.fecha, servicios.tipo  
FROM trabajadores, servicios  
WHERE trabajadores.DNI=servicios.DNI;
```

Observa aquí como se indica en la cláusula FROM las dos tablas de las que extraemos datos. Es importante que te fijas también en como se unen ambas tablas igualando en la cláusula WHERE el campo de unión de ambas tablas, que en el ejemplo es el DNI.

Ejemplo: “Visualizar todos los servicios. Interesa que aparezca el nombre del trabajador que hizo el servicio, el tipo de servicio y el nombre del cliente al que se le hizo el servicio”

```
SELECT trabajadores.nombre, servicios.tipo, clientes.nombre  
FROM trabajadores, servicios, clientes  
WHERE trabajadores.DNI=servicios.DNI AND clientes.CIF=servicios.CIF;
```

Observa aquí una consulta sobre tres tablas, las cuales aparecen en el FROM. Es necesario indicar en la cláusula WHERE los campos de unión. La tabla Trabajadores se relaciona con la tabla Servicios a través del campo DNI, y la tabla Trabajadores se relaciona con Clientes a través del campo CIF. Observa el uso de AND para unir varias condiciones.

Ejemplo: “Visualizar los servicios que hayan costado más de 200 euros. Interesa ver la fecha del servicio, el nombre del cliente y el coste ordenado por cantidad”

```
SELECT servicios.fecha, clientes.nombre, servicios.cantidad  
FROM servicios, clientes  
WHERE servicios.CIF=clientes.CIF AND servicios.cantidad>200  
ORDER BY servicios.cantidad;
```

Observa como la cláusula WHERE contiene por un lado la condición de unión de ambas tablas y por otro lado la condición que se busca (cantidad > 200)

FORMA DE INDICAR CRITERIOS EN LA CLÁUSULA WHERE

Se van a indicar a continuación una serie de reglas que se deben seguir a la hora de crear condiciones en la cláusula WHERE de una consulta SQL

Operadores Relacionales

Operador	Significa...	Ejemplos
=	Igual que	WHERE cantidad = 200 WHERE tipo = 'Limpieza' WHERE fecha = #8-5-2006#
>	Mayor que (para números) Posterior a (para fechas)	WHERE cantidad > 200 WHERE fecha > #8-5-2006#
>=	Mayor o igual que (para números) Esa fecha o posterior (para fechas)	WHERE cantidad >= 200 WHERE fecha >= #8-5-2006#
<	Menor que (para números) Anterior a (para fechas)	WHERE cantidad < 200 WHERE fecha < #8-5-2006#
<=	Menor o igual que (para números) Esa fecha o anterior (para fechas)	WHERE cantidad <= 200 WHERE fecha <= #8-5-2006#
<>	Distinto de (para fechas, números y textos)	WHERE cantidad <> 200 WHERE fecha <> #8-5-2006# WHERE tipo <> 'Limpieza'
Between...and	Entre valor1 y valor2 (aplicable a números y fechas)	WHERE cantidad BETWEEN 100 AND 200 WHERE fecha BETWEEN #8-5-2006# AND #1-12-2006#
Like 'cadena*'	Que empiece por <i>cadena</i> (aplicable a textos)	WHERE nombre LIKE 'Jose*'

Like '*cadena'	Que termine por <i>cadena</i> (aplicable a textos)	WHERE nombre LIKE '*Jose'
Like '*cadena*'	Que contenga <i>cadena</i> (aplicable a textos)	WHERE nombre LIKE '*Jose*'
IS NULL	Que el campo esté vacío (aplicable a números, textos, fechas)	WHERE telefono IS NULL
NOT ... IS NULL	Que el campo no esté vacío (aplicable a números, textos, fechas)	WHERE NOT telefono IS NULL

Operadores Lógicos

Operador	Significa...	Ejemplos
AND	Obliga a que se cumplan las dos condiciones que une.	WHERE cantidad > 200 AND tipo = 'Limpieza' (Debe cumplirse que la cantidad sea mayor de 200 y que el tipo de servicio sea <i>Limpieza</i>)
OR	Basta con que se cumpla una sola de las dos condiciones que une.	WHERE cantidad > 200 OR tipo = 'Limpieza' (Basta con que la cantidad sea mayor de 200, o que el tipo de servicio sea <i>Limpieza</i> para que se cumpla la condición)
NOT	Si no se cumple la condición, la condición global se cumple.	WHERE NOT cantidad > 200 (Se cumple la condición si la cantidad NO es mayor de 200)

Forma de indicar los valores

Como puedes observar en los ejemplos anteriores, tendrás que tener en cuenta las siguientes reglas para indicar valores:

Valores numéricos

Indica los valores numéricos tal cual, teniendo en cuenta que debes usar el punto decimal cuando quieras representar decimales.

Ejemplo:

WHERE cantidad > 200.12

Valores de texto

Los valores de texto se indican rodeándolos entre comillas simples: ‘

Ejemplos:

WHERE nombre = ‘Jose’

WHERE dirección LIKE ‘*avenida*’

Valores de fecha

Las fechas se indican rodeándolas entre almohadillas #. Se debe tener en cuenta que las fechas deben indicarse separadas por guiones – o barras / y que su formato debe ser el siguiente:

Mes – Día – Año

Ejemplos:

WHERE fecha > #02-01-2005#

(Significa que la fecha debe ser posterior al 1 de febrero de 2005)

WHERE fecha <> #10-12-2006#

(Significa que la fecha debe ser distinta del 12 de Octubre de 2006)

Forma de indicar los campos

Normalmente los campos que se usan en el WHERE (y en otras cláusulas) se indican de la siguiente forma:

Tabla.Campo

Por ejemplo,

WHERE trabajadores.sueldo > 1000

(*Sueldo* es un campo de la tabla *trabajadores*)

Si tenemos la seguridad de que no existe otro campo de otra tabla que se llame igual, entonces podemos prescindir del nombre de la tabla.

Por ejemplo,

WHERE sueldo > 1000

(No existe otro campo sueldo en otras tablas de la consulta)

En el caso de que el nombre del campo contenga espacios, entonces tendremos que rodear el campo con corchetes.

Por ejemplo,

```
WHERE [sueldo del trabajador] > 1000
```

(El campo se llama *sueldo del trabajador*)

Alta de registros en SQL

Se empezará estudiando como añadir nuevos registros en una base de datos usando el lenguaje SQL.

Código base en SQL para añadir nuevos registros

Para añadir un registro en una tabla se usa la siguiente sintaxis:

```
INSERT INTO tabla  
VALUES (valor1, valor2, valor3, ..., valor n);
```

En la cláusula INSERT INTO se indica la tabla en la que se quiere introducir una nueva fila (registro), y en la cláusula VALUES se indican los valores de la fila que se quiere insertar. Estos valores tienen que estar indicados en el orden en el que están definidos los campos en la tabla correspondiente.

Por ejemplo:

```
INSERT INTO trabajadores  
VALUES ('30.234.234-A', 'María', 'López', 1250.45, #01/02/2006, '4455-RSD');
```

En esta instrucción se está introduciendo un nuevo trabajador en la tabla trabajadores. Concretamente se está introduciendo un trabajador con las siguientes características:

DNI: 30.234.234-A
Nombre: María
Apellidos: López
Sueldo: 1250,45
Fecha de entrada: 02/01/2006
Matrícula: 4455-RSD

Los valores están indicados en el mismo orden en el que están los campos en la tabla: DNI, Nombre, Apellidos, Sueldo, Fecha, Matrícula.

Si se quiere introducir un nuevo registro, pero indicando solo los valores de algunos campos, se puede usar entonces esta otra sintaxis para la instrucción INSERT INTO:

```
INSERT INTO tabla  
(campo a, campo b, campo c)  
VALUES (valor del campo a, valor del campo b, valor del campo c);
```

En este caso solo se introducen los valores correspondientes a tres campos, el resto de los campos del registro se quedarán vacíos.

Por ejemplo:

```
INSERT INTO trabajadores  
(DNI, Apellidos, Sueldo)  
VALUES ('30.234.234-A', 'López', 1250.45);
```

En este caso introducimos a un trabajador de apellido López, con DNI 30.234.234-A que cobra 1250,45 euros. Tanto la matrícula de su coche como su nombre y la fecha quedan en blanco.

Formato de los valores.

Recuerda que debes seguir ciertas reglas para introducir valores:

Las cadenas se escriben entre comillas simples. Por ejemplo: 'López'

Se usa el punto decimal en los números reales: Por ejemplo: 1250.45

Las fechas se indicarán entre almohadillas # y hay que indicar primero el mes, luego el día y finalmente el año. Por ejemplo: #12-20-2007# es 20 de diciembre de 2007.

En el caso de querer especificar explícitamente que un campo esté vacío, se puede indicar el valor NULL.

Ejemplo:

```
INSERT INTO trabajadores  
(DNI, Apellidos, Sueldo)  
VALUES ('30.234.234-A', NULL, 1250.45);
```

Los apellidos del trabajador están vacíos en este ejemplo.

Modificación de registros en SQL

Es posible usar el lenguaje SQL para modificar los datos de una tabla. Se puede incluso modificar los datos de aquellos registros que cumplan una condición en concreto.

Código base en SQL para modificar registros

Para modificar los datos de los registros de una tabla se usará el siguiente código general:

```
UPDATE tabla a modificar  
SET campo1 = nuevovalor1, campo2 = nuevovalor2, ..., campon =  
nuevovalorn  
WHERE condición;
```

En la cláusula UPDATE se indica la tabla cuyos registros se van a modificar.

La cláusula SET permite indicar los cambios que se realizarán. Se debe indicar el campo que se va a cambiar y el nuevo valor que se introducirá en el campo. Como puede observar, se pueden indicar varios campos a modificar.

La cláusula WHERE permite indicar una condición. Esto hará que solo se cambien los valores de aquellos registros que cumplan la condición. La cláusula WHERE es opcional, y si no se indicara se cambiarían todos los registros de la tabla.

Ejemplo:

```
UPDATE trabajadores  
SET sueldo = 1200, matricula='3355-AAA'  
WHERE fecha < #01/01/2004#;
```

En este ejemplo se les asigna un sueldo de 1200 euros y el coche con matrícula 3355-AAA a todos aquellos trabajadores que hayan entrado en la empresa antes del 1-1-2004.

Ejemplo:

```
UPDATE trabajadores  
SET sueldo = 1300;
```

En este ejemplo, se les asigna a todos los trabajadores un sueldo de 1300 euros ya que no se ha indicado cláusula WHERE.

Si se quisiera hacer un cambio puntual a un registro en concreto, tendremos que hacer uso de su campo clave para indicar la condición. Recuerda que el campo clave es el que identifica de forma única a un registro de la tabla.

Por ejemplo:

```
UPDATE trabajadores  
SET sueldo = 1300
```

```
WHERE DNI='33.444.333-A';
```

En este ejemplo se estoy asignando un sueldo de 1300 al trabajador con DNI 33.444.333-A, y a ningún otro (ya que se supone que no habrá otro con dicho DNI)

Eliminación de registros en SQL

Al igual que podemos añadir nuevos registros (filas) a las tablas usando SQL, también podemos usar este lenguaje para eliminar registros de las tablas.

Código base en SQL para eliminar registros

Para eliminar registros de una tabla se usará el siguiente código general:

```
DELETE FROM tabla de la que se quiere eliminar  
WHERE condición de los registros que se eliminarán;
```

En la cláusula DELETE FROM se indica la tabla de la que eliminaremos registros. En la cláusula WHERE se indica la condición que deben cumplir los registros que eliminaremos.

Por ejemplo:

```
DELETE FROM trabajadores  
WHERE sueldo>1000;
```

En este ejemplo se están eliminando de la tabla *trabajadores* aquellos trabajadores cuyo *sueldo* sea superior a 1000.

Si se quiere eliminar un solo registro de la tabla, será necesario hacer referencia a su campo clave. Recuerda que el campo clave es el que identifica de forma única a cada registro.

Por ejemplo, si queremos eliminar un trabajador en concreto, indicaremos su DNI en la condición:

```
DELETE FROM trabajadores  
WHERE DNI='33.444.555-A';
```

Esta instrucción SQL borra al trabajador con DNI 33.444.555-A (solamente a él, ya que se supone que no habrá otro trabajador que tenga ese mismo DNI)

En la cláusula WHERE de una instrucción DELETE, las condiciones se indican tal como se vio en la hoja anterior para las instrucciones SELECT.

Si se quiere eliminar todo el contenido de una tabla, se puede usar una instrucción DELETE sin indicar ninguna condición. Por ejemplo:

```
DELETE FROM trabajadores;
```

Esta instrucción eliminaría todos los registros de la tabla *trabajadores*.

Acceso a Base de Datos desde una aplicación Java

Pasos Generales para preparar una Aplicación Java para acceder a una Base de Datos

Para preparar nuestra aplicación Java para que pueda acceder a una Base de Datos, es necesario realizar tres pasos:

1. Cargar el controlador de la base de datos.

El *controlador* define el tipo de base de datos que se va a usar (base de datos de Access, o de MySQL, o de cualquier otro gestor de base de datos)

En nuestro caso, tendremos que indicar el controlador para base de datos de Access.

2. Crear un objeto conexión (*Connection*)

Para crear este objeto hay que indicar la situación del fichero de base de datos, el usuario y la contraseña de dicha base de datos. El objeto conexión abre el fichero de la base de datos.

3. Crear un objeto sentencia (*Statement*)

El objeto sentencia se crea a partir del objeto conexión anterior. Los objetos sentencia permiten realizar acciones sobre la base de datos usando instrucciones SQL.

Es decir, a través del objeto sentencia introduciremos datos en la base de datos, eliminaremos datos, haremos modificaciones, y extraeremos datos de la base de datos.

Así pues, este objeto es vital. Este objeto es el que realmente permite el acceso a los datos de la base de datos y la manipulación de dicha base de datos.

EJERCICIO 1:

Crear una aplicación de bases de datos que permita mostrar los datos de los trabajadores almacenados en la base de datos MANEMPSA.



1. Para poder acceder y manipular una base de datos, es necesario tener dos objetos:
 - Un objeto del tipo **Connection**, al que llamaremos *conexion*. Este objeto define la conexión con la base de datos.
 - Un objeto del tipo **Statement**, al que llamaremos *sentencia*. Este objeto permite manipular la base de datos.
 - Una vez definidos los objetos *conexión* y *sentencia*, necesarios para el acceso a la base de datos, prepararemos nuestro programa para que pueda acceder a la base de datos MANEMPSA.MDB. Esto se hará en el constructor. En primer lugar, añada al constructor una llamada a una función **PrepararBaseDatos**.

```
Connection conexion;  
Statement sentencia;  
public AccesoBD()  
{  
    initComponents();  
    PrepararBaseDatos();  
}
```

2. Crea el método *PrepararBaseDatos* debajo del constructor y empieza a programar lo siguiente:

```

void PrepararBaseDatos(){

//1.- Cargar el controlador
    try
    {
        String controlador="com.mysql.jdbc.Driver";
        Class.forName(controlador).newInstance();

    } catch (Exception ex)
    {
        JOptionPane.showMessageDialog(null,"Error al cargar el
controlador.");
    }

//2.- Crear el objeto conexión.

    String DBURL="jdbc:mysql://localhost/manempsa";
    String usuario="root";
    String password="";
    try
    {

conexion=DriverManager.getConnection(DBURL,usuario,password);
    } catch (SQLException ex)
    {

Logger.getLogger(AccesoBD.class.getName()).log(Level.SEVERE, null,
ex);
    }
    try
    {

//3.- Crear el objeto sentencia.

sentencia=conexion.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,Res
ultSet.CONCUR_READ_ONLY);
    } catch (SQLException ex)
    {

Logger.getLogger(AccesoBD.class.getName()).log(Level.SEVERE, null,
ex);
    }

}

```

El método **PrepararBaseDatos** siempre será igual, solo cambiará el nombre de la base de datos a usar.

El objetivo de todo este código de preparación para el acceso al fichero de la base de datos es obtener un objeto llamado *sentencia* que nos posibilitará la manipulación de los datos de la base de datos, usando órdenes SQL.

3. Ahora vamos a **realizar consultas SQL** usando el objeto sentencia. Cuando se pulse el botón *Ver Datos de Trabajadores* tendremos que extraer los datos de la tabla *trabajadores* para poder mostrarlos. Para ello, escribe el siguiente código dentro del evento *actionPerformed* del botón *btnVerDatos*:

```
private void btnVerDatosActionPerformed(java.awt.event.ActionEvent
evt)//GEN-FIRST:event_btnVerDatosActionPerformed

{ //GEN-HEADEREND:event_btnVerDatosActionPerformed

    String info="";
    double totalsu=0;
    try
    {
        // TODO add your handling code here:
        ResultSet r=sentencia.executeQuery("select * from trabajadores
order by nombre");
        r.beforeFirst();
        while(r.next())
        {
            info=info+r.getString("nombre")+
+r.getString("apellido")+"\t\t "+r.getString("sueldo")+"\n";
            totalsu=totalsu+r.getDouble("sueldo");
        }
        JOptionPane.showMessageDialog(null, info);
        JOptionPane.showMessageDialog(null, "La suma de los
suealdoses: "+totalsu);

    } catch (SQLException ex)
    {
        JOptionPane.showMessageDialog(null,"Error al consultar la
tabla trabajadores"+ ex);
    }
}
```

El bucle que acabas de programar es un código “clásico” para manipular un **ResultSet**. Siempre que quieras recorrer todas las filas del *ResultSet* harás algo como esto:

```
r.beforeFirst();
while (r.next()) {
    manipulación de la fila
}
```

LOS OBJETOS RESULTSET

Debes imaginarte el objeto *ResultSet* *r* como una tabla que contiene el resultado de la consulta SQL que se ha ejecutado. En nuestro caso, la consulta SQL que hemos ejecutado ha extraído toda la tabla *trabajadores*. Por tanto nuestro *ResultSet* contiene toda la tabla *trabajadores*.

El objeto *r* por tanto podría representarse así:

Trabajadores					
DNI	Nombre	Apellidos	Sueldo	Fecha	Matricula
BOF					
21.123.123-A	Ana	Ruiz	1200	02/03/2002	3322-ASR
22.333.444-C	Francisco	López	1000	01/06/2006	1144-BBB
12.321.567-B	Juan	Pérez	1120	04/05/2002	4433-ABB
EOF					

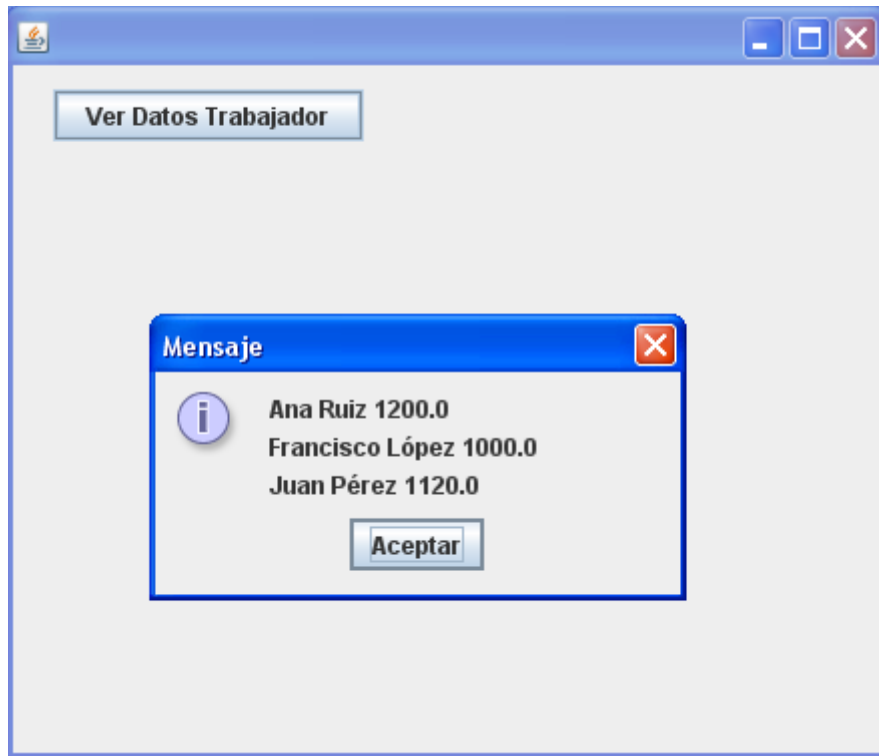
La fila BOF significa “comienzo de fichero” y representa una fila anterior al primer registro del *ResultSet*.

La fila EOF significa “final de fichero” y representa una fila posterior al último registro del *ResultSet*.

La flecha indica la posición actual donde estamos situados dentro de la tabla del *ResultSet*.

Los objetos *ResultSet* poseen diversos métodos para cambiar la posición actual en la tabla del *ResultSet*. Dicho de otro modo: “para mover la flecha”. Veamos algunos de estos métodos (se supone que el objeto *ResultSet* se llama *r*):

r.next();	<input type="checkbox"/> Mueve la flecha a la siguiente fila
r.previous();	<input type="checkbox"/> Mueve la flecha a la fila anterior
r.first();	<input type="checkbox"/> Mueve la flecha a la primera fila
r.last();	<input type="checkbox"/> Mueve la flecha a la última fila
r.beforeFirst()	<input type="checkbox"/> Mueve la flecha a la fila BOF
r.afterLast()	<input type="checkbox"/> Mueve la flecha a la fila EOF
r.absolute(n)	<input type="checkbox"/> Mueve la flecha a la fila <i>n</i> del <i>ResultSet</i> . Las filas se empiezan a numerar por 1.



Al pulsar el botón...

Aparecen los datos de todos los trabajadores de la base de datos.

4. Una vez finalizado el programa, es una buena costumbre cerrar la base de datos que estamos manejando. Esto se hace cerrando la “conexión” con la base de datos.

Para hacer esto se usa el método *close* del objeto *conexión*.

Esto se hará en el momento en que se finalice el programa, es decir, en el evento *windowClosing* de la ventana principal:

```
private void formWindowClosing(java.awt.event.WindowEvent evt) {  
    // TODO: Agrega su código aquí:  
    try {  
        conexion.close();  
    } catch (Exception e) {  
        JOptionPane.showMessageDialog(null, "No se pudo cerrar la base de datos");  
    }  
}
```

CONCLUSIÓN

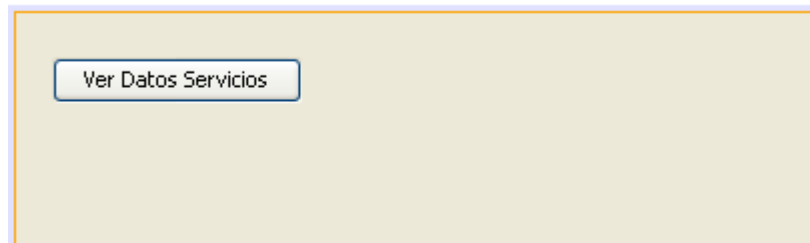
Para crear un programa Java que pueda usar una base de datos será necesario realizar los siguientes pasos:

- Colocar el fichero de base de datos en una subcarpeta de la carpeta del proyecto java.
- Preparar el acceso a la base de datos (en el constructor)
 - * Se crearán dos objetos: *conexión (Connection)* y *sentencia (Statement)*
 - * Se cargará el controlador del tipo de base de datos a usar
 - * Se creará el objeto conexión indicando el fichero de la base de datos.
 - * Se creará el objeto sentencia a partir del objeto conexión
- Se usará el objeto *sentencia* para ejecutar consultas SQL en la base de datos.
- Las consultas SQL ejecutadas en la base de datos se almacenan en objetos del tipo *ResultSet*
- Un objeto *ResultSet* tiene forma de tabla conteniendo el resultado de la consulta SQL
 - * Los objetos *ResultSet* tienen métodos para seleccionar el registro de la tabla
 - * Los objetos *ResultSet* tienen métodos que permiten extraer el dato de un campo en concreto.

EJERCICIO 2:

Realizar una pequeña aplicación de base de datos que nos muestre información sobre los servicios realizados en la empresa MANEMPESA.

Se pretende simplemente que al pulsar el botón *btnServicios* aparezcan en un JOptionPane datos sobre los servicios almacenados en la base de datos.



En la parte superior de la ventana añade un botón con el texto *Ver Datos Servicios* que se llame *btnServicios*.

1. Preparar el proyecto para que permita el acceso a la base de datos MANEMPESA.MDB .

Solo tiene que añadir el siguiente código a su ventana principal:

```
public class ventanaprincipal extends javax.swing.JFrame {  
  
    Connection conexion;  
    Statement sentencia;  
  
    /** Creates new form ventanaprincipal */  
    public ventanaprincipal() {  
        initComponents();  
        PrepararBaseDatos();  
    }  
}
```

Declara los objetos
globales *conexion* y
sentencia

Haz la llamada al método
PrepararBaseDatos

El método ***PrepararBaseDatos*** siempre será igual, solo tienes que indicar el nombre de la base de datos a usar.

2. Programar el botón para visualizar los servicios. Entra dentro del ***actionPerformed*** de este botón y programa lo siguiente:


```

private void btnServiciosActionPerformed(java.awt.event.ActionEvent evt) {
// TODO: Agrega su código aquí:

String info="";

try {
    ResultSet r=sentencia.executeQuery("select * from servicios order by cantidad");

    r.beforeFirst();
    while (r.next()) {
        info=info+r.getString("tipo")+" "+r.getString("cantidad")+"\n";
    }
    JOptionPane.showMessageDialog(null,info);

} catch(Exception e) {
    JOptionPane.showMessageDialog(null,"Error al consultar la tabla servicios");
}

}

```

El código, lo que hace es ejecutar la consulta SQL

```
select * from servicios order by cantidad
```

la cual extrae todos los servicios almacenados en la tabla *servicios* ordenados por *cantidad* de menor a mayor.

El resultado de esta consulta se almacena en un *ResultSet* y se usa un bucle típico que recorre el *ResultSet* y muestra el tipo de cada servicio y la cantidad:

```

while (r.next()) {
    info=info+r.getString("tipo")+" "+r.getString("cantidad")+"\n";
}

```

3. Añadamos el cierre de la conexión de la base de datos en el *windowClosing* de nuestra ventana:

```

private void formWindowClosing(java.awt.event.WindowEvent evt) {
// TODO: Agrega su código aquí:
    try {
        conexion.close();
    } catch(Exception e) {
        JOptionPane.showMessageDialog(null,"No se pudo cerrar la base de datos");
    }

}

```

4. Mejorar el programa de forma que se muestre de cada servicio el tipo, la cantidad y la fecha en que se hizo. Por tanto, haz el siguiente cambio en el código del *actionPerformed* del botón *btnServicios*:

```

private void btnServiciosActionPerformed(java.awt.event.ActionEvent evt) {
    TODO: Agrega su código aquí:

    String info="";

    try {
        ResultSet r=sentencia.executeQuery("select * from servicios order by cantidad");

        r.beforeFirst();
        while (r.next()) {
            info=info+r.getString("fecha")+" "+r.getString("tipo")+" "+
                r.getString("cantidad")+"\n";
        }
        JOptionPane.showMessageDialog(null,info);

    } catch (Exception e) {
        JOptionPane.showMessageDialog(null,"Error al consultar la tabla servicios");
    }

}

```

Como se ha podido observar, las fechas extraídas del *ResultSet* tienen un formato distinto al que usamos normalmente. Para que la fecha aparezca en un formato español haz los siguientes cambios en el código:

```

private void btnServiciosActionPerformed(java.awt.event.ActionEvent evt) {
// TODO: Agregue su código aquí:
    String info="";
    String cadfecha; //cadena para fechas
    String caddia; //cadena para el día
    String cadmes; //cadena para el mes
    String cadanio; //cadena para el año

    try {
        ResultSet r=sentencia.executeQuery("select * from servicios order by cantidad");

        r.beforeFirst();
        while (r.next()) {

            cadfecha=r.getString("fecha");
            cadanio=cadfecha.substring(0,4);
            cadmes=cadfecha.substring(5,7);
            caddia=cadfecha.substring(8,10);
            cadfecha=caddia+"/"+cadmes+"/"+cadanio;

            info=info+cadfecha+" "+r.getString("tipo")+" "+
                r.getString("cantidad")+"\n";
        }
        JOptionPane.showMessageDialog(null,info);

    } catch(Exception e) {
        JOptionPane.showMessageDialog(null,"Error al consultar la tabla servicios");
    }
}

```



Observa como ahora las fechas aparecen correctamente...

5. Ahora mejoraremos el programa para que los costes de los servicios aparezcan con coma decimal, en vez de punto decimal:



Haremos que aquí aparezcan comas, en vez de puntos...

Modifica el código de la siguiente forma:

```

private void btnServiciosActionPerformed(java.awt.event.ActionEvent evt) {
// TODO: Agregue su código aquí:
String info="";
String cadfecha; //cadena para fechas
String caddia; //cadena para el día
String cadmes; //cadena para el mes
String cadanio; //cadena para el año
String cadcoste; //cadena para el coste►

try {
    ResultSet r=sentencia.executeQuery("select * from servicios order by cantidad");

    r.beforeFirst();
    while (r.next()) {
        cadfecha=r.getString("fecha");
        cadanio=cadfecha.substring(0,4);
        cadmes=cadfecha.substring(5,7);
        caddia=cadfecha.substring(8,10);
        cadfecha=caddia+"/"+cadmes+"/"+cadanio;

        cadcoste=r.getString("cantidad");
        cadcoste=cadcoste.replace(".",",");

        info=info+cadfecha+" "+r.getString("tipo")+" "+cadcoste+"\n";
    }
    JOptionPane.showMessageDialog(null,info);

} catch (Exception e) {
    JOptionPane.showMessageDialog(null,"Error al consultar la tabla servicios");
}
}

```

Se ha añadido una variable de cadena llamada *cadcoste* que almacenará el coste de cada servicio.

En el código del bucle, recogemos la cantidad en dicha variable y luego usamos el método de cadena *replace* para reemplazar los puntos por comas:

```

cadcoste=r.getString("cantidad");
cadcoste=cadcoste.replace(".",",");

```

Finalmente, mostramos la cadena de coste en la concatenación:

```

info=info+cadfecha+" "+r.getString("tipo")+" "+cadcoste+"\n";

```



VALORES NULOS

Es posible que algún campo de algún registro de la tabla esté vacío. Es decir, que sea nulo. Si esto ocurre, entonces al extraer dicho dato de la tabla usando *getString* aparecerá el valor *null* en el *JOptionPane*.

6. Para comprobar esta circunstancia, agrega un nuevo botón a la ventana principal con el texto “Ver Datos de Clientes”. Llámalo por ejemplo *btnClientes*. Al pulsar este botón aparecerá el listado de clientes de la empresa. Concretamente debe aparecer el nombre del cliente, el teléfono 1 y el teléfono 2. Para ello añade el siguiente código dentro del evento *actionPerformed* del botón.

```

private void btnClientesActionPerformed(java.awt.event.ActionEvent evt) {
// TODO: Agregue su código aquí:
String info="";
String nomcli; //cadena el nombre de cliente
String tfno1; //cadena para telefono 1
String tfno2; //cadena para telefono 2

try {
    ResultSet r=sentencia.executeQuery("select * from clientes order by nombre");

    r.beforeFirst();
    while (r.next()) {
        nomcli=r.getString("nombre");
        tfno1=r.getString("tfno1");
        tfno2=r.getString("tfno2");

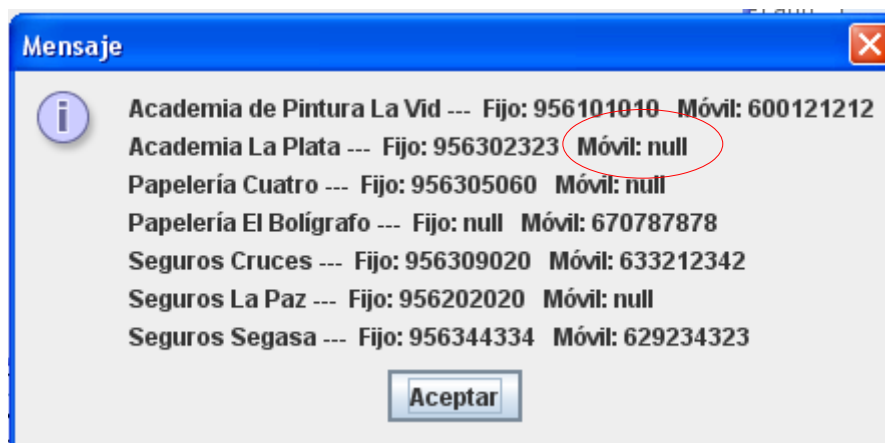
        info=info+nomcli+" --- "+"Fijo: "+tfno1+" "+"Móvil: "+tfno2+"\n";
    }
    JOptionPane.showMessageDialog(null,info);

} catch (Exception e) {
    JOptionPane.showMessageDialog(null,"Error al consultar la tabla clientes");
}
}

```

Este código es prácticamente igual que el anterior. Simplemente ejecuta una consulta SQL usando el objeto *sentencia* que permite extraer el contenido de la tabla *clientes*, y luego recorre el *ResultSet* mostrando los campos *nombre*, *teléfono 1* y *teléfono 2* en un *JOptionPane*.

Ejecuta el programa ahora y prueba a pulsar este nuevo botón. Observa el resultado. Cada vez que un cliente no tenga un teléfono, aparecerá el valor "null" en el *JOptionPane*:



7. Vamos a arreglar esto de forma que aparezca el texto “no tiene” en vez de la cadena “null”. Modifica el código como se indica:

```
private void btnClientesActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO: Agrega su código aquí:  
    String info="";  
    String nomcli; //cadena el nombre de cliente  
    String tfno1; //cadena para telefono 1  
    String tfno2; //cadena para telefono 2  
  
    try {  
        ResultSet r=sentencia.executeQuery("select * from clientes order by nombre");  
  
        r.beforeFirst();  
        while (r.next()) {  
            nomcli=r.getString("nombre");  
            tfno1=r.getString("tfno1");  
            if (tfno1==null) {  
                tfno1="no tiene";  
            }  
            tfno2=r.getString("tfno2");  
            if (tfno2==null) {  
                tfno2="no tiene";  
            }  
  
            info=info+nomcli+" --- Fijo: "+tfno1+" Móvil: "+tfno2+"\n";  
        }  
        JOptionPane.showMessageDialog(null,info);  
  
    } catch (Exception e) {  
        JOptionPane.showMessageDialog(null,"Error al consultar la tabla clientes");  
    }  
}
```

Como puedes ver, lo que se hace ahora es comprobar si el valor extraído del *ResultSet* es null, y en ese caso, se concatena la cadena *no tiene*. En caso contrario se concatena el valor del campo.

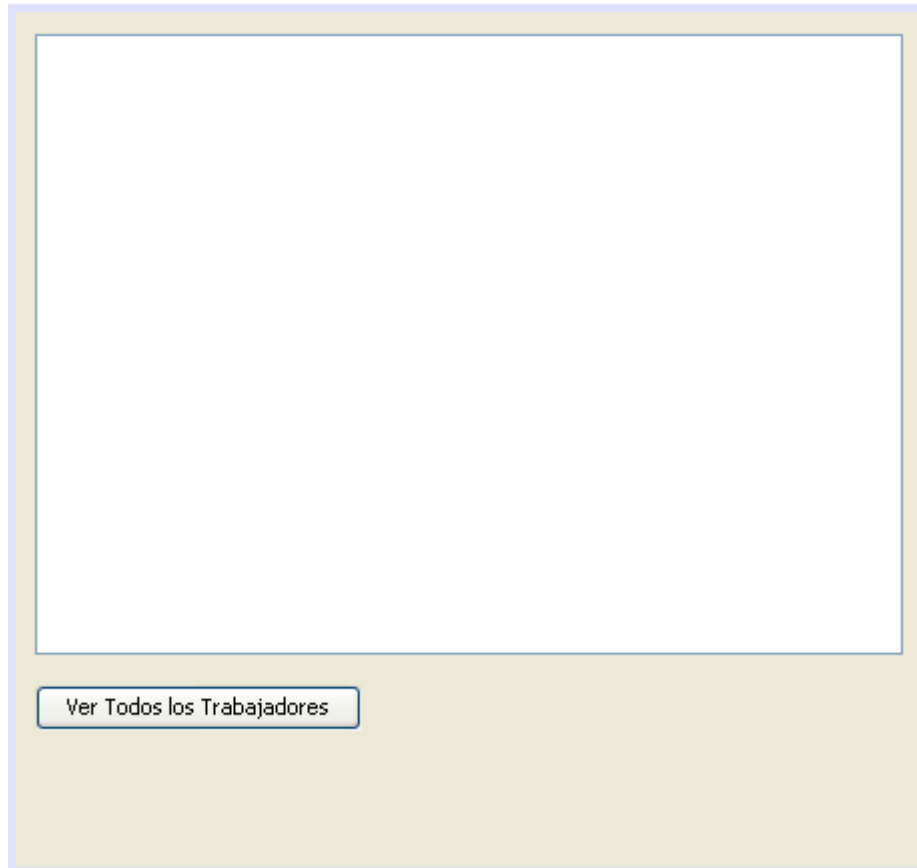


EJERCICIO 3:

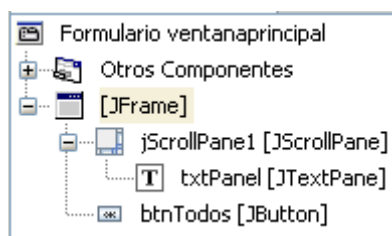
Se quiere realizar una aplicación de base de datos que nos muestre información sobre los trabajadores de la empresa MANEMPESA.

Esta aplicación le dará al usuario la capacidad de elegir la información que quiere extraer de la base de datos. Es decir, el usuario tendrá cierto control sobre las consultas que se realicen.

1. Añade a la ventana un JTextPane y un botón de momento:



El botón se llamará *btnTodos* y el JTextPane se llamará *txtPanel*.



2. Para que este programa pueda trabajar con la base de datos MANEMPESA tendrá que prepararlo haciendo lo siguiente:

- a. Añadir al programa los objetos *conexión (Connection)* y *sentencia (Statement)* como globales.
- b. Crear el procedimiento *PrepararBaseDatos* y llamarlo desde el constructor.
- c. Cerrar la conexión desde el evento *windowClosing*

Realiza estos tres pasos que se han indicado antes de continuar.

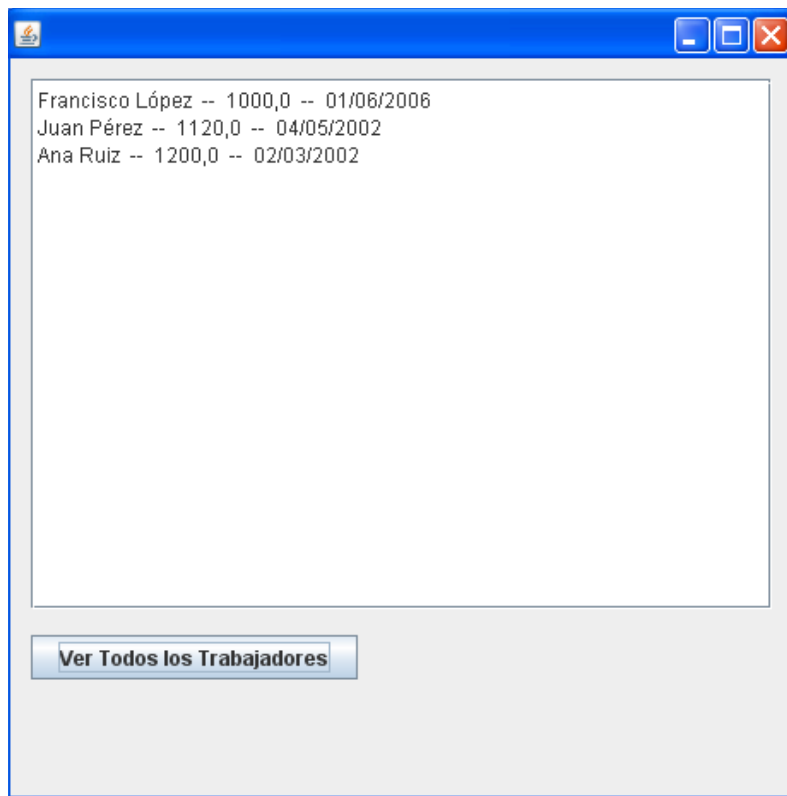
3. Ya se puede programar el botón *btnTodos*. Se pretende que al pulsar este botón aparezca en el panel *txtPanel* el contenido de la tabla *trabajadores*. Para ello, programa el siguiente código dentro del *actionPerformed* del botón *btnTodos*:

```
private void btnTodosActionPerformed(java.awt.event.ActionEvent evt) {
// TODO: Agregue su código aquí:
String info="";
String cadnombre; //cadena para el nombre
String cadapell; //cadena para los apellidos
String cadsueldo; //cadena para el sueldo
String cadfecha; //cadena para la fecha

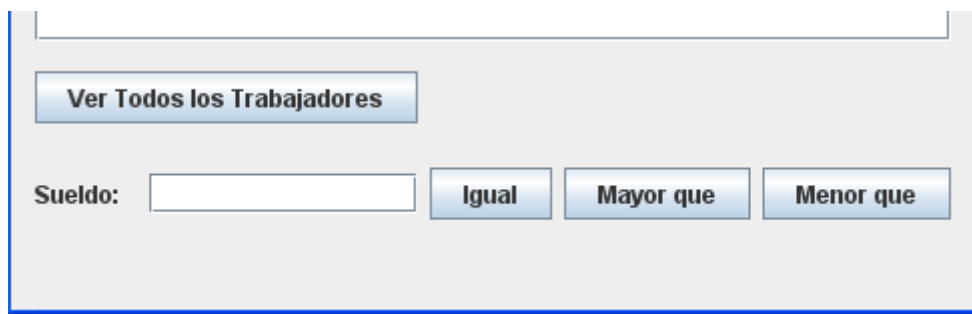
try {
    ResultSet r=sentencia.executeQuery("select * from trabajadores order by sueldo");

    r.beforeFirst();
    while (r.next()) {
        cadnombre=r.getString("nombre");
        cadapell=r.getString("apellidos");
        cadsueldo=r.getString("sueldo");
        cadsueldo = cadsueldo.replace(".",",");
        cadfecha=r.getString("fecha");
        cadfecha=cadfecha.substring(8,10)+"/"+cadfecha.substring(5,7)+"/"+
            cadfecha.substring(0,4);
        info=info+cadnombre+" "+cadapell+" -- "+cadsueldo+" -- "+cadfecha+"\n";
    }
    txtPanel.setText(""); //vacío el panel de texto
    txtPanel.setText(info); //meto la cadena info

} catch (Exception e) {
    JOptionPane.showMessageDialog(null,"Error al consultar la tabla trabajadores");
}
}
```



4. Se va a mejorar el programa. Añadir un cuadro de texto llamado *txtSueldo* y luego tres botones llamados respectivamente *btnMayor*, *btnMenor* y *btnIgual*. La parte inferior de la ventana quedará así:



Se pretende que estos botones funcionen de la siguiente forma:

- El usuario introducirá un sueldo en el cuadro de texto *txtSueldo*.
- Luego, si pulsa el botón *Igual*, aparecerá en el panel todos los trabajadores que tengan un sueldo igual al introducido.
- En cambio, si pulsa el botón *Mayor*, aparecerá en el panel todos los trabajadores que tengan un sueldo mayor que el introducido.
- Y si pulsa el botón *Menor*, aparecerá en el panel todos los trabajadores que tengan un sueldo menor que el introducido.

Se empezará programando el botón *Igual*.

Programar en el *actionPerformed* del botón *btnIgual* lo siguiente. (Nota: El código siguiente es prácticamente igual al anterior, solo se hace un pequeño cambio. Puede copiar y pegar y luego hacer la modificación que se indica)

```
private void btnIgualActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO: Agregue su código aquí:  
    String info="";  
    String cadnombre; //cadena para el nombre  
    String cadapell; //cadena para los apellidos  
    String cadsueldo; //cadena para el sueldo  
    String cadfecha; //cadena para la fecha  
  
    try {  
        String consulta;  
        consulta = "select * from trabajadores where sueldo = "+txtSueldo.getText();  
        ResultSet r=sentencia.executeQuery(consulta);  
  
        r.beforeFirst();  
        while (r.next()) {  
            cadnombre=r.getString("nombre");  
            cadapell=r.getString("apellidos");  
            cadsueldo=r.getString("sueldo");  
            cadsueldo = cadsueldo.replace(".",",");  
            cadfecha=r.getString("fecha");  
            cadfecha=cadfecha.substring(8,10)+"/"+cadfecha.substring(5,7)+"/"+  
                cadfecha.substring(0,4);  
            info=info+cadnombre+" "+cadapell+" -- "+cadsueldo+" -- "+cadfecha+"\n";  
        }  
        txtPanel.setText(""); //vacío el panel de texto  
        txtPanel.setText(info); //meto la cadena info  
  
    } catch (Exception e) {  
        JOptionPane.showMessageDialog(null,"Error al consultar la tabla trabajadores");  
    }  
}
```

Aquí se crea una variable de texto llamada *consulta* y luego se concatena en ella la cadena:

`select * from trabajadores where sueldo =`

con

lo que contenga el cuadro de texto sueldo – es decir `txtSueldo.getText()`

Si el cuadro de texto del sueldo contuviera un 1000, entonces la cadena resultante sería:

`select * from trabajadores where sueldo = 1000`

Es decir, se construye una consulta que busca los sueldos de 1000 euros.

Para probar, escribe el valor 1000 en el cuadro de texto y luego pulsar el botón *Igual*. El resultado será que aparecen solo los trabajadores que tengan 1000 de sueldo.

The screenshot shows a Java Swing window with a blue title bar and standard window controls. Inside, there is a text area at the top containing the text "Francisco López -- 1000,0 -- 01/06/2006". Below the text area is a button labeled "Ver Todos los Trabajadores". At the bottom, there is a label "Sueldo:" followed by a text input field containing the value "1000". To the right of the input field are three buttons: "Igual", "Mayor que", and "Menor que".

Al pulsar el botón *Igual* se construye una consulta usando el contenido del cuadro de texto *txtSueldo*.

Al ejecutarse la consulta se muestran los trabajadores de 1000 euros.

Programar ahora el botón *Mayor que* de la siguiente forma:

```

private void btnMayorActionPerformed(java.awt.event.ActionEvent evt) {
// TODO: Agregue su código aquí:
    String info="";
    String cadnombre; //cadena para el nombre
    String cadapell; //cadena para los apellidos
    String cadsueldo; //cadena para el sueldo
    String cadfecha; //cadena para la fecha

    try {
        String consulta;
        consulta = "select * from trabajadores where sueldo > "+txtSueldo.getText();
        ResultSet r=sentencia.executeQuery(consulta);

        r.beforeFirst();
        while (r.next()) {
            cadnombre=r.getString("nombre");
            cadapell=r.getString("apellidos");
            cadsueldo=r.getString("sueldo");
            cadsueldo = cadsueldo.replace(".",",");
            cadfecha=r.getString("fecha");
            cadfecha=cadfecha.substring(8,10)+"/"+cadfecha.substring(5,7)+"/"+
                cadfecha.substring(0,4);
            info=info+cadnombre+" "+cadapell+" -- "+cadsueldo+" -- "+cadfecha+"\n";
        }
        txtPanel.setText(""); //vacío el panel de texto
        txtPanel.setText(info); //meto la cadena info

    } catch (Exception e) {
        JOptionPane.showMessageDialog(null,"Error al consultar la tabla trabajadores");
    }
}

```

Como se puede observar, el código es igual. Simplemente cambia el operador en la cadena que se concatena. Ahora se usa un *mayor que*.

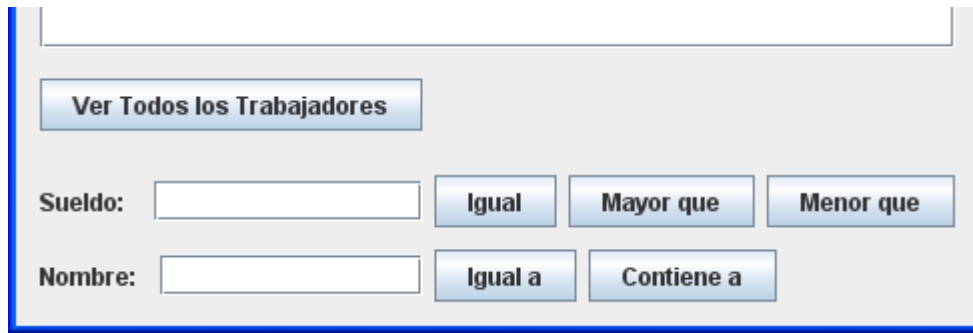
Es decir, si el usuario introdujera un 1000 en el cuadro de texto del sueldo, el resultado de la concatenación en la variable consulta sería la siguiente cadena:

```
select * from trabajadores where sueldo > 1000
```

Para probar el programa introduce el valor 1000 en el sueldo y luego pulsando el botón *Mayor que*. El resultado será que aparece en el panel de texto los trabajadores que cobran más de 1000 euros.

Programar ahora el botón *Menor de manera similar*.

5. Vamos a seguir mejorando el programa. Añadir ahora el siguiente cuadro de texto y los siguientes botones:



The image shows a graphical user interface for a program. At the top, there is a button labeled "Ver Todos los Trabajadores". Below this, there are two filter sections. The first section is for salary, labeled "Sueldo:", and contains a text input box followed by three buttons: "Igual", "Mayor que", and "Menor que". The second section is for name, labeled "Nombre:", and contains a text input box followed by two buttons: "Igual a" and "Contiene a".

El cuadro de texto se llamará *txtNombre* mientras que el botón "*Igual a* " se llamará *btnNombreIgual* y el botón "*Contiene a*" se llamará *btnContiene*.

Estos botones funcionarán de la siguiente forma:

- El usuario introducirá un nombre en el cuadro de texto *txtNombre*.
- Si luego pulsa el botón *Igual a*, entonces aparecerán todos aquellos trabajadores que tengan exactamente dicho nombre.
- Si en cambio pulsa el botón *Contiene a*, entonces aparecerán todos aquellos trabajadores cuyo nombre contenga la palabra que se haya escrito en el cuadro de texto.

Programación del botón igual:

```

private void btnNombreIgualActionPerformed(java.awt.event.ActionEvent evt) {
// TODO: Agregue su código aquí:
String info="";
String cadnombre; //cadena para el nombre
String cadapell; //cadena para los apellidos
String cadsueldo; //cadena para el sueldo
String cadfecha; //cadena para la fecha
try {
String consulta;
consulta = "select * from trabajadores where nombre = '"+txtNombre.getText()+"'";
ResultSet r=sentencia.executeQuery(consulta);

r.beforeFirst();
while (r.next()) {
cadnombre=r.getString("nombre");
cadapell=r.getString("apellidos");
cadsueldo=r.getString("sueldo");
cadsueldo = cadsueldo.replace(".",",");
cadfecha=r.getString("fecha");
cadfecha=cadfecha.substring(8,10)+"/"+cadfecha.substring(5,7)+"/"+
cadfecha.substring(0,4);
info=info+cadnombre+" "+cadapell+" -- "+cadsueldo+" -- "+cadfecha+"\n";
}
txtPanel.setText(""); //vacío el panel de texto
txtPanel.setText(info); //meto la cadena info

} catch (Exception e) {
JOptionPane.showMessageDialog(null,"Error al consultar la tabla trabajadores");
}
}

```

La consulta SQL se consigue concatenando tres cadenas (se han puesto en color para facilitar la comprensión):

Primera cadena: `select * from trabajadores where nombre = '`

Segunda cadena: `lo que contenga el cuadro de texto:`
`txtNombre.getText()`

Tercera cadena: `'`

Supongamos que el cuadro de texto contiene la palabra *Ana*, el resultado de la concatenación sería:

`select * from trabajadores where nombre = 'Ana'`

Es decir, el resultado de la concatenación sería una consulta SQL que muestra aquellos trabajadores que tengan de nombre Ana.

Programar el botón **Contiene a** de forma que el usuario escriba un texto en el cuadro del nombre, y al pulsar el botón **Contiene a** aparezcan todos aquellos trabajadores cuyo nombre contenga el texto escrito.

Por ejemplo, si el usuario introduce el texto “an” en el cuadro, al pulsar **Contiene a** aparecerán los trabajadores que se llamen: **Juan, Antonio, Antonia, Manolo, Ana**, etc (todos contienen el texto ‘an’)

Para ello programar lo siguiente en el *actionPerformed* del botón **Contiene a**.

```
private void btnContieneActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO: Agregue su código aquí:  
    String info="";  
    String cadnombre; //cadena para el nombre  
    String cadapell; //cadena para los apellidos  
    String cadsueldo; //cadena para el sueldo  
    String cadfecha; //cadena para la fecha  
    try {  
        String consulta;  
        consulta = "select * from trabajadores where nombre like '%" + txtNombre.getText() + "%'";  
        ResultSet r=sentencia.executeQuery(consulta);  
  
        r.beforeFirst();  
        while (r.next()) {  
            cadnombre=r.getString("nombre");  
            cadapell=r.getString("apellidos");  
            cadsueldo=r.getString("sueldo");  
            cadsueldo = cadsueldo.replace(".",",");  
            cadfecha=r.getString("fecha");  
            cadfecha=cadfecha.substring(8,10)+"/"+cadfecha.substring(5,7)+"/"+  
                cadfecha.substring(0,4);  
            info=info+cadnombre+" "+cadapell+" -- "+cadsueldo+" -- "+cadfecha+"\n";  
        }  
        txtPanel.setText(""); //vacío el panel de texto  
        txtPanel.setText(info); //meto la cadena info  
    } catch (Exception e) {  
        JOptionPane.showMessageDialog(null,"Error al consultar la tabla trabajadores");  
    }  
}
```

En este caso, para crear la consulta se concatenan tres cadenas:

Primera cadena: `select * from trabajadores where nombre like`
`'%`

Segunda cadena: `lo que contenga el cuadro de texto: txtNombre.getText()`

Tercera cadena: `%'`

Supongamos que escribimos en el cuadro de texto del nombre la palabra *an*, el resultado de la concatenación sería el siguiente:

```
select * from trabajadores where nombre like '%an%'
```

La condición *nombre like '%an%'* significa que contenga la palabra *an*.

6. Sigamos mejorando el programa. Añade estos cuadros de texto y estos botones:

The diagram shows a user interface for filtering workers. It includes a button "Ver Todos los Trabajadores". Below it are three rows of input fields and comparison buttons:

- Sueldo:** A text input field followed by three buttons: "Igual", "Mayor que", and "Menor que".
- Nombre:** A text input field followed by two buttons: "Igual a" and "Contiene a".
- Fecha:** Three separate text input fields (for day, month, and year) separated by hyphens, followed by two buttons: "Anterior" and "Después".

Arrows point from labels below to specific UI elements:

- txtDia** points to the first input field of the "Fecha" row.
- txtMes** points to the second input field of the "Fecha" row.
- txtAnio** points to the third input field of the "Fecha" row.
- btnAnterior** points to the "Anterior" button.
- btnDespues** points to the "Después" button.

El objetivo de estos elementos añadidos es el siguiente:

- El usuario introducirá una fecha (día, mes y año) en los cuadros de texto *txtDia*, *txtMes*, *txtAño*.
- Luego, si pulsa el botón *Anterior*, aparecerán los trabajadores que hayan entrado en la empresa antes de la fecha indicada.
- Si pulsa el botón *Después*, en cambio, aparecerán los trabajadores que hayan entrado en la empresa después de la fecha indicada.

Empezaremos programando el botón **Anterior**. Accede a su *actionPerformed* e incluye el siguiente código (es un código similar al anterior, solo tienes que realizar un pequeño cambio):

```
private void btnAnteriorActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO: Agrega su código aquí:  
    String info="";  
    String cadnombre; //cadena para el nombre  
    String cadapell; //cadena para los apellidos  
    String cadsueldo; //cadena para el sueldo  
    String cadfecha; //cadena para la fecha  
    try {  
        String consulta;  
        consulta = "select * from trabajadores where fecha < #";  
        consulta = consulta + txtMes.getText()+"-"+txtDia.getText()+"-"+txtAño.getText();  
        consulta = consulta + "#";  
        ResultSet r=sentencia.executeQuery(consulta);  
  
        r.beforeFirst();  
        while (r.next()) {  
            cadnombre=r.getString("nombre");  
            cadapell=r.getString("apellidos");  
            cadsueldo=r.getString("sueldo");  
            cadsueldo = cadsueldo.replace(".",",");  
            cadfecha=r.getString("fecha");  
            cadfecha=cadfecha.substring(8,10)+"/"+cadfecha.substring(5,7)+"/"+  
                cadfecha.substring(0,4);  
            info=info+cadnombre+" "+cadapell+" -- "+cadsueldo+" -- "+cadfecha+"\n";  
        }  
        txtPanel.setText(""); //vacío el panel de texto  
        txtPanel.setText(info); //meto la cadena info  
    } catch (Exception e) {  
        JOptionPane.showMessageDialog(null,"Error al consultar la tabla trabajadores");  
    }  
}
```

Presta mucha atención al código remarcado. En él, se construye una consulta que permite mostrar aquellos trabajadores cuya fecha de entrada en la empresa sea anterior a la indicada en los cuadros de texto.

Esto se consigue concatenando varias cadenas:

Primera cadena: `select * from trabajadores where fecha < #`
Segunda cadena: *lo que contiene el cuadro de texto del mes: txtMes.getText()*
Tercera cadena: `/`
Cuarta cadena: *lo que contiene el cuadro de texto del día: txtDia.getText()*
Quinta cadena: `/`
Sexta cadena: *lo que contiene el cuadro de texto del año: txtAnio.getText()*
Séptima cadena: `#`

Por ejemplo, supongamos que:

- en el cuadro *txtDia* se introdujo un 20.
- en el cuadro *txtMes* se introdujo un 12.
- En el cuadro *txtAnio* se introdujo un 2005.

Entonces, la cadena resultante de la concatenación será:

```
select * from trabajadores where fecha < #12/20/2006#
```

Es decir, la cadena resultante es una consulta SQL que busca los trabajadores cuya fecha de entrada en la empresa sea anterior al 20 del 12 del 2006

Si se ejecuta el programa, se introduce una fecha en las casillas correspondientes y luego se pulsa el botón *Anterior*, aparecen los trabajadores que entraron antes de la fecha indicada.

The screenshot shows a Windows application window with a blue title bar. Inside, there's a list box containing two entries: "Ana Ruiz -- 1200,0 -- 02/03/2002" and "Juan Pérez -- 1120,0 -- 04/05/2002". Below the list box is a button labeled "Ver Todos los Trabajadores". Underneath that are three input fields: "Sueldo:", "Nombre:", and "Fecha:". The "Sueldo:" field has three buttons: "Igual", "Mayor que", and "Menor que". The "Nombre:" field has two buttons: "Igual a" and "Contiene a". The "Fecha:" field has three input boxes (containing "1", "7", and "2003") and two buttons: "Anterior" and "Después".

Introduce un día, mes y año en los cuadros correspondientes.

Al pulsar el botón *Anterior*, el programa mostrará los trabajadores que hayan entrado antes de esa fecha.

De manera similar se debe programar el botón *Después*. Al pulsar este botón, se mostrarán los trabajadores que hayan entrado antes de la fecha indicada en los cuadros de texto. El código es prácticamente igual al código del botón *Anterior*.