



**CICLO: [DAM]**  
**MÓDULO DE [PROGRAMACIÓN]**

# **[Tarea N° 09]**

**Alumno:**  
**[Juan Carlos Filter Martín]**  
**[15456141A]**

## Contenido

<b>1. Documentos que se adjuntan a este informe.....</b>	<b>3</b>
<b>2. RA09_b) Se han programado conexiones con bases de datos.....</b>	<b>3</b>
D) Crear una BD MySQL con el nombre equivalente al DNI.....	3
E) Contendrá una tabla llamada Departamentos con los siguientes campos:.....	4
Código tipo numérico entero (primary key).....	5
Nombre tipo String de 50 caracteres.....	5
Id de localización tipo numérico entero.....	5
Id del mánager tipo numérico entero.....	5
Tabla Departamentos creada (Resultado).....	5
F) Crear usuario para acceder a la BD:.....	6
G) Creamos el proyecto Java agregándole las librerías necesarias para trabajar con BD.....	7
<b>3. RA09_d) Se han creado programas para recuperar y mostrar información almacenada en bases de datos.....</b>	<b>8</b>
A) Crear la interfaz de usuario.....	8
Insertar.....	9
Clase GestorBD.....	9
JFrame form (Interfaz de usuario).....	11
Resultado Interfaz Gráfica.....	12
Resultado (phpMyAdmin).....	14
<b>4. RA09_e) Se han efectuado borrados y modificaciones sobre la información almacenada.....</b>	<b>14</b>
Borrar.....	14
Clase GestorBD.....	15
JFrame Form (Interfaz de usuario).....	19
Resultado (Interfaz Gráfica).....	20
Resultado (phpMyAdmin).....	21
Modificar.....	22
Clase GestorBD.....	22
JFrame form (Interfaz de usuario).....	24
Resultado (Interfaz Gráfica).....	25
Resultado (phpMyAdmin).....	26
<b>5. RA09_e) Se han efectuado borrados y modificaciones sobre la información almacenada.....</b>	<b>27</b>
Mostrar.....	27
Clase GestorBD.....	27
JFrame form (Interfaz de usuario).....	29
Resultado (Interfaz Gráfica).....	30
Limpiar Campos y salir.....	31

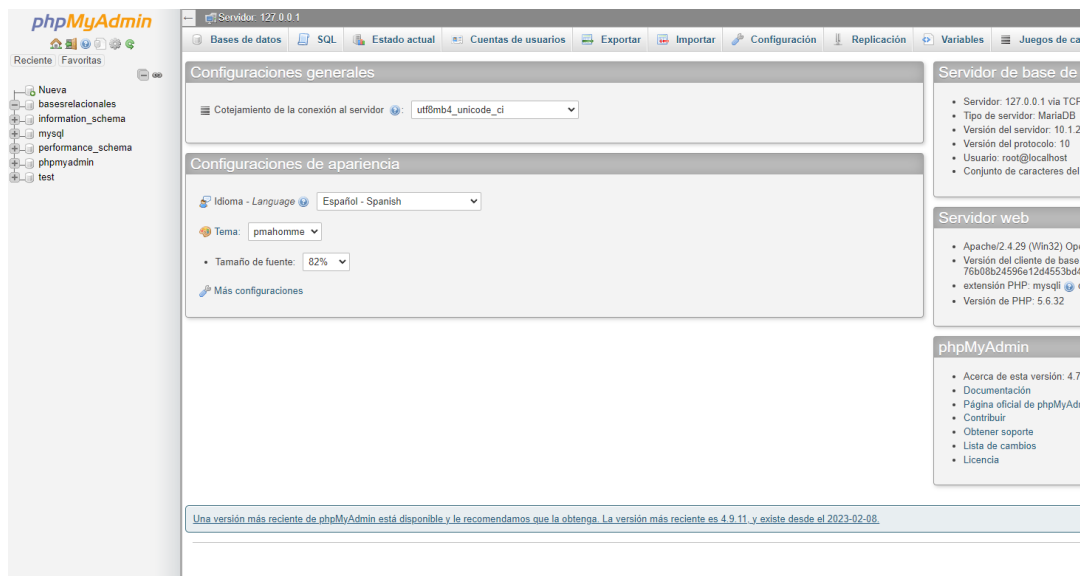
## 1. Documentos que se adjuntan a este informe.

A continuación se detallan los documentos que componen la presente entrega de la tarea:

1. Informe de elaboración de la tarea.
2. Proyecto Java
3. Script.sql exportado de creación de la BD

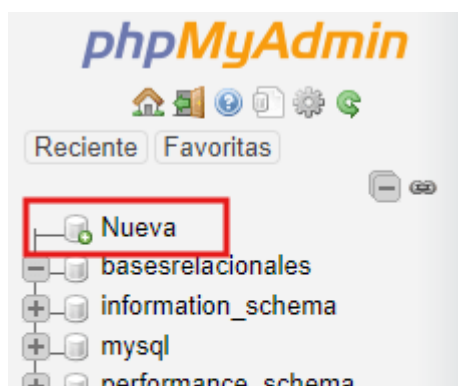
## 2. RA09\_b) Se han programado conexiones con bases de datos.

Desde XAMPP iniciamos el servicio Apache y MySQL. Entramos a phpMyAdmin.



### ***D) Crear una BD MySQL con el nombre equivalente al DNI***

Desde el panel lateral izquierdo creamos una BD nueva.



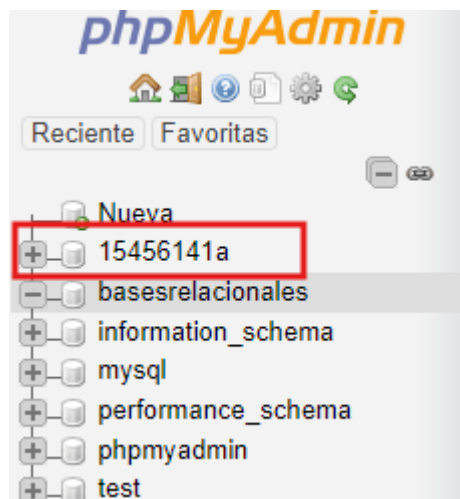
Con el nombre del DNI: 15456141A y pulsamos en crear

Crear base de datos

15456141A utf8mb4\_general\_ci **Crear**

Base de datos	Cotejamiento	Acción
<input type="checkbox"/> basesrelacionales	latin1_swedish_ci	Seleccionar privilegios
<input type="checkbox"/> information_schema	utf8_general_ci	Seleccionar privilegios
<input type="checkbox"/> mysql	latin1_swedish_ci	Seleccionar privilegios
<input type="checkbox"/> performance_schema	utf8_general_ci	Seleccionar privilegios

Y tendríamos la BD creada



**E) Contendrá una tabla llamada Departamentos con los siguientes campos:**

Creamos la tabla Departamentos con 4 columnas ya va a contener 4 campos

Crear tabla

Nombre: Departamentos Número de columnas: 4

✖ **Código tipo numérico entero (primary key)**

Seleccionar desde las columnas centrales

INT

Ninguno

☐

PRIMARY

☐

PRIMARY

✖ **Nombre tipo String de 50 caracteres**

Seleccionar desde las columnas centrales

VARCHAR

50

Ninguno

☐

---

☐

✖ **Id de localización tipo numérico entero**

Seleccionar desde las columnas centrales

INT

Ninguno

☐

---

☐

✖ **Id del mánager tipo numérico entero**

Seleccionar desde las columnas centrales

INT

Ninguno

☐

---

☐

✖ **Tabla Departamentos creada (Resultado)**

#	Nombre	Tipo	Cotejamiento	Atributos	Nulo	Predeterminado	Comentarios	Extra	Acción
<input type="checkbox"/>	1	codigo			No	Ninguna			Cambiar  Eliminar  Primari
<input type="checkbox"/>	2	nombre	utf8mb4_general_ci		No	Ninguna			Cambiar  Eliminar  Primari
<input type="checkbox"/>	3	id_localizacion			No	Ninguna			Cambiar  Eliminar  Primari
<input type="checkbox"/>	4	id_manager			No	Ninguna			Cambiar  Eliminar  Primari

## F) Crear usuario para acceder a la BD:

Crearemos un usuario para poder acceder a la BD y asignamos todos los privilegios para poder realizar las diferentes consultas, insert, etc.

phpMyAdmin

Reciente Favoritas

Nueva 15456141a

Nueva departamentos

Columnas

Nueva codigo

id\_localizacion

id\_manager

nombre

Indice

Nuevo PRIMARY

basesrelacionales

information\_schema

mysql

performance\_schema

phpmyadmin

test

Servidor: 127.0.0.1

Bases de datos SQL Estado actual Cuentas de usuarios Exportar Importar Configuración Replicación Varias

### Agregar cuenta de usuario

**Información de la cuenta**

Nombre de usuario: Use el campo de texto:

Nombre de Host: Cualquier servidor %

Contraseña: Use el campo de texto:  Strength: Extremadamente débil

Debe volver a escribir:

Complemento de autenticación: Autenticación de MySQL nativo

Generar contraseña:

**Base de datos para la cuenta de usuario**

☐ Crear base de datos con el mismo nombre y otorgar todos los privilegios.

☐ Otorgar todos los privilegios al nombre que contiene comodín (username\_%).

☒ Otorgar todos los privilegios para la base de datos 15456141a.

**Privilegios globales** ☒ Seleccionar todo

Nota: Los nombres de los privilegios de MySQL están expresados en inglés.

**Datos**

- ☒ SELECT
- ☒ INSERT
- ☒ UPDATE
- ☒ DELETE
- ☒ FILE

**Estructura**

- ☒ CREATE
- ☒ ALTER
- ☒ INDEX
- ☒ DROP
- ☒ CREATE TEMPORARY TABLES
- ☒ SHOW VIEW
- ☒ CREATE ROUTINE
- ☒ ALTER ROUTINE
- ☒ EXECUTE
- ☒ CREATE VIEW
- ☒ EVENT
- ☒ TRIGGER

**Administración**

- ☒ GRANT
- ☒ SUPER
- ☒ PROCESS
- ☒ RELOAD
- ☒ SHUTDOWN
- ☒ SHOW DATABASES
- ☒ LOCK TABLES
- ☒ REFERENCES
- ☒ REPLICATION CLIENT
- ☒ REPLICATION SLAVE
- ☒ CREATE USER

**Limites de recursos**

Nota: si cambia los parámetros de estas opciones a 0 (cero), remueve el límite.

MAX QUERIES PER HOUR

MAX UPDATES PER HOUR

MAX CONNECTIONS PER HOUR

MAX USER\_CONNECTIONS

✓ Ha agregado un nuevo usuario.

```
CREATE USER 'jcfm'@'%' IDENTIFIED VIA mysql_native_password USING '****';GRANT ALL PRIVILEGES ON *.* TO 'jcfm'
```

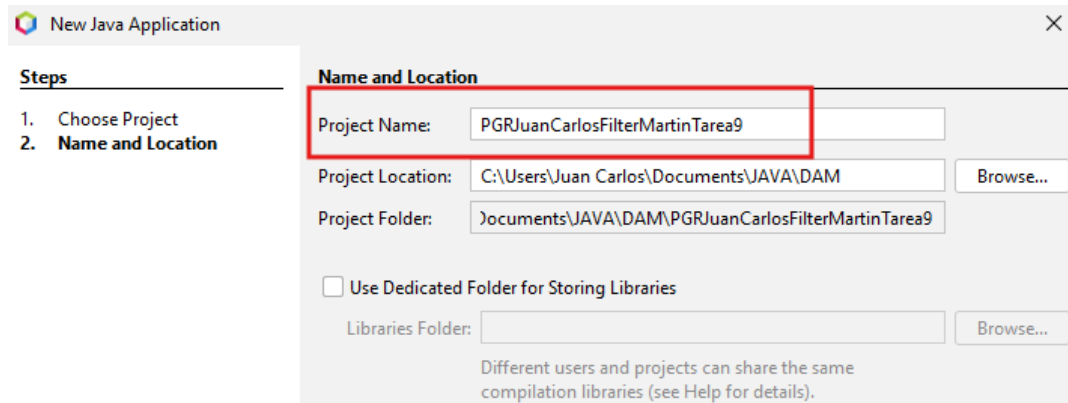
Base de datos

Tabla

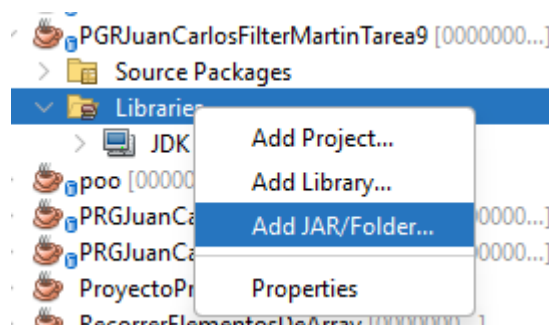
Rutina

## G) Creamos el proyecto Java agregándole las librerías necesarias para trabajar con BD

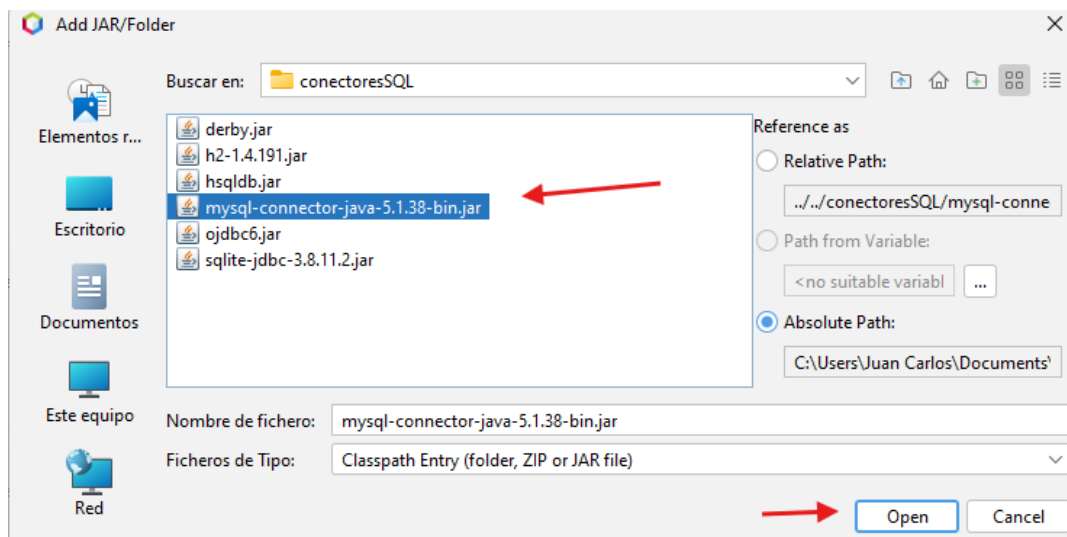
1. Creamos el proyecto



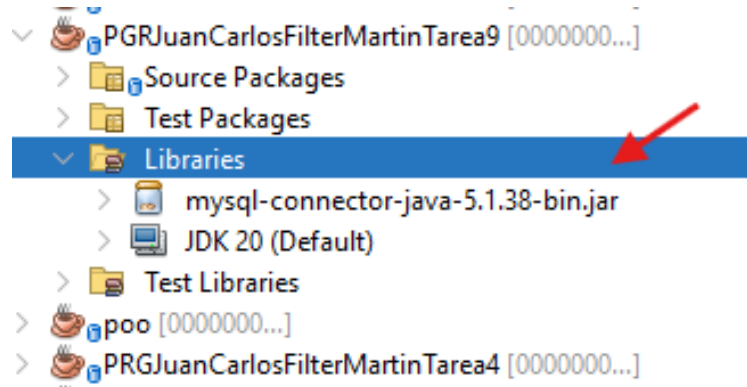
2. Agregamos librerías pulsando botón derecho sobre libraries > **Add JAR/Folder**



3. Vamos al directorio donde se encuentre la librería “conectoresSQL.zip” y pulsamos en **Open**



4. Y tendríamos las librerías agregadas al proyecto.



3. RA09\_d) Se han creado programas para recuperar y mostrar información almacenada en bases de datos.

### ***A) Crear la interfaz de usuario***

Tendremos que crear una clase JFrame Form y empezar a diseñar la interfaz gráfica.

Datos del Departamento		Mostrar datos del departamento			
Código	<input type="text"/>	Código	Nombre	ID Localización	ID Manager
Nombre	<input type="text"/>				
ID Localización	<input type="text"/>				
ID Manager	<input type="text"/>				
<input type="button" value="INSERTAR"/>		<input type="button" value="LIMPIAR CAMPOS"/>		<input type="button" value="MOSTRAR"/>	
<input type="button" value="MODIFICAR"/>		<input type="button" value="BORRAR"/>		<input type="button" value="SALIR"/>	

(Con la interfaz construida vamos a crear los diferentes eventos asociados a cada botón)



## × Insertar

Se va a recoger la información introducida por el usuario en (código, nombre, id\_localizacion, id\_manager).

Con los datos recogidos se va a acceder a la BD e insertar el nuevo departamento en la tabla departamentos.

Una vez insertado el departamento se mostrarán todos los registros en la Tabla.

### Clase GestorBD

```
//Metodo para insertar Departamentos
public void alta(int codigo, String nombre, int id_localizacion, int id_manager) {

    try {
        conectar();
        String resultado;

        //Variable que almacena el INSERT INTO
        //Se le dice a java que en un punto dado las "?" deben ser sustituidas
        String sql = "INSERT INTO Departamentos VALUES (?, ?, ?, ?)";

        //Se Prepara el statement mandandole la variable que contiene el INSERT INTO
        preparedStatement = connection.prepareStatement(string: sql);

        ////Indicamos cada posicion del placeholder y a que hace referencia
        preparedStatement.setInt(i: 1, i1: codigo);          //1º ?
        preparedStatement.setString(i: 2, string: nombre);  // 2º ?
        preparedStatement.setInt(i: 3, i1: id_localizacion); // 3º ?
        preparedStatement.setInt(i: 4, i1: id_manager);    // 4º ?

        //Se realiza el insert into
        preparedStatement.executeUpdate();
        resultado = "Insertado";
        InfoshowDialog(msg: resultado);

    } catch (SQLException e) {
        InfoshowDialog("Error al insertar en el departamento: " + e.getMessage());
    } finally {
        desconectar();
    }
}
```

- ✖ Se conectará a la BD con el método `conectar()`;

```
//Metodo para conectar con la BD
public void conectar() {
    try {
        //Decir que tipo de bbdd tengo
        Class.forName(className: "com.mysql.jdbc.Driver");
        //Se crea la conexión (url, usuario, pwd)
        connection = DriverManager.getConnection(url:"jdbc:mysql://localhost/15456141a", user: "jcfm", password: "jcfm");

    } catch (ClassNotFoundException | SQLException e) {
        JOptionPane.showMessageDialog(parentComponent:null, "Error de conexión: " + e.getMessage(),
            title: "Error", messageType:JOptionPane.ERROR_MESSAGE);
    }
}
```

Carga el driver JDBC.

Y se crea una conexión en el que se va a indicar donde se encuentra la BD y el login

```
//Metodo para insertar Departamentos
public void alta(int codigo, String nombre, int id_localizacion, int id_manager) {

    try {
        conectar();
        String resultado;

        //Variable que almacena el INSERT INTO
        //Se le dice a java que en un punto dado las "?" deben ser sustituidas
        String sql = "INSERT INTO Departamentos VALUES (?, ?, ?, ?)";

        //Se Prepara el statement
        preparedStatement = connection.prepareStatement(string: sql);

        preparedStatement.setInt(i: 1, il: codigo);          //1º ?
        preparedStatement.setString(i: 2, string: nombre);   // 2º ?
        preparedStatement.setInt(i: 3, il: id_localizacion); // 3º ?
        preparedStatement.setInt(i: 4, il: id_manager);      // 4º ?

        //Se realiza el insert into
        preparedStatement.executeUpdate();
        resultado = "Insertado";
        InfoshowDialog(msg:resultado);

    } catch (SQLException e) {
        InfoshowDialog("Error al insertar en el departamento: " + e.getMessage());
    } finally {
        desconectar();
    }
}
```

- ✖ Almacenamos el INSERT INTO en una variable
- ✖ Preparamos el statement en el que le pasamos el INSERT INTO y más adelante insertamos en los placeholder (indicándolo con la posición "?" + el parámetro código, nombre, id\_localización, id\_manager)
- ✖ Ejecutamos la sentencia mediante la variable `preparedStatement` junto al método `executeUpdate()`

- ✖ **Desconectamos la BD** al pasar al finally.

```
//Metodo para desconectar con la BD
public void desconectar() {
    try {
        //En caso de que Connection, preparedStatement o ResultSet no sea null se desconecta con la BD
        if (connection != null) {
            connection.close();
        }
        if (prepareStatement != null) {
            preparedStatement.close();
        }
        if (resultSet != null) {
            resultSet.close();
        }
    } catch (SQLException e) {
        InfoshowDialog("Error de desconexión: " + e.getMessage());
    }
}
```

Si connection no es null > cerramos la conexión

Si preparedStatement no es null > cerramos la conexión

Si resultSet no es null > cerramos la conexión

## **[Frame form (Interfaz de usuario)]**

- ✖ Ahora recogeremos los datos que introduzca el usuario y llamaremos a los diferentes métodos ya creados para realizar las diferentes funciones

```
private void insertarBtActionPerformed(java.awt.event.ActionEvent evt) {
    GestorBD baseDatos = new GestorBD();

    String msg = CamposVaciosJFrame.controlarCamposVacios(codigo: this.codigoTf.getText(), nombre: this.nombreTf.getText(),
        id_localizacion: this.idlocalizacionTf.getText(), id_manager: this.idmanagerTf.getText());

    if (msg == null) {
        //llamada del metodo alta de departamento
        baseDatos.alta(codigo: Integer.parseInt(s: this.codigoTf.getText()), nombre: this.nombreTf.getText(),
            id_localizacion: Integer.parseInt(s: this.idlocalizacionTf.getText()), id_manager: Integer.parseInt(s: this.idmanagerTf.getText()));

        //Llamada del metodo mostrar
        mostrar();
        //Limpiamos los campos TextField
        limpiarCampos();
    } else {
        baseDatos.ErrorshowDialog(msg);
    }
}
```

- ✖ Creamos un **objeto de la clase GestorBD**.
- ✖ **Llamada al método estático controlarCamposVacios** de la clase CamposVaciosJFrame (controlará que no se deje los campos vacíos a la ahora de introducir un departamento)
- ✖ **Si devuelve null** significa que no hay campos TextField en blanco entonces **pasamos a llamar al método de altas**.
- ✖ **Llamamos al método mostrar** y este mostrará todos los departamentos que contiene la BD en la tabla de la interfaz gráfica.

## Resultado Interfaz Gráfica

- ✖ Introducimos los datos, pulsamos en INSERTAR

**Datos del Departamento**

Código: 5

Nombre: Logística

ID Localización: 10002

ID Manager: 150

INSERTAR MODIFICAR BORRAR

**Mostrar datos del departamento**

Código	Nombre	ID Localización	ID Manager
1	Finanzas	25	10
2	Comercial	70021	92
3	Directivo	5772	58
4	Compras	233	34

LIMPIAR CAMPOS MOSTRAR SALIR

- ✖ Aparecerá una ventana emergente indicando que se ha insertado.

The screenshot shows a web application interface with two main panels. The left panel, titled 'Datos del Departamento', contains four input fields: 'Código' (with value 5), 'Nombre' (with value Logística), 'ID Localización', and 'ID Manager'. Below these fields are three buttons: 'INSERTAR', 'MODIFICAR', and 'BORRAR'. The right panel, titled 'Mostrar datos del departamento', contains a table with the following data:

Código	Nombre	ID Localización	ID Manager
1	Finanzas	25	10
2	Comercial	70021	92
3	Directivo	5772	58
4	Compras	233	34

Below the table are three buttons: 'LIMPIAR CAMPOS', 'MOSTRAR', and 'SALIR'. A red rectangular box highlights a modal dialog box that appears over the 'Nombre' field. The dialog box is titled 'ÉXITO' and contains an information icon, the text 'Insertado', and an 'OK' button.

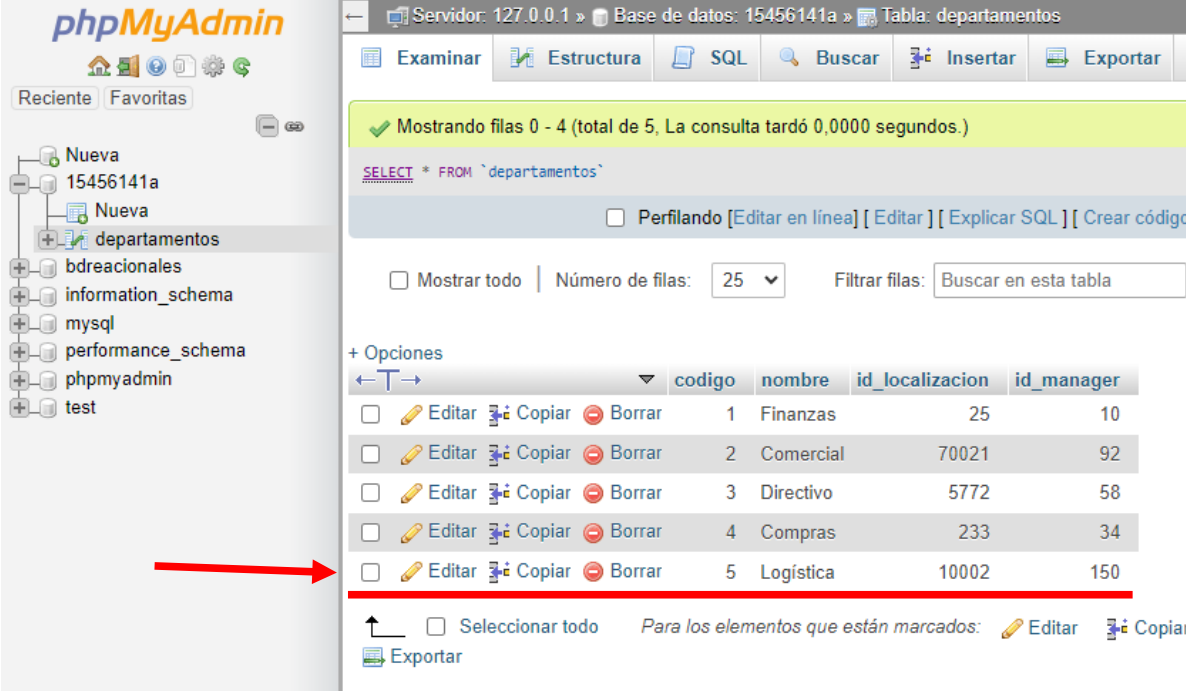
- ✖ Aparecerá la lista de los departamentos que hay en la BD con el nuevo departamento insertado en la tabla.

The screenshot shows the same web application interface as before, but the modal dialog box is no longer present. The 'Datos del Departamento' panel now has empty input fields for 'Código', 'Nombre', 'ID Localización', and 'ID Manager'. The 'Mostrar datos del departamento' panel shows the same table as before, but with an additional row at the bottom, which is highlighted with a red horizontal line:

Código	Nombre	ID Localización	ID Manager
1	Finanzas	25	10
2	Comercial	70021	92
3	Directivo	5772	58
4	Compras	233	34
5	Logística	10002	150

The buttons in both panels remain the same as in the previous screenshot.

## Resultado (phpMyAdmin)



The screenshot shows the phpMyAdmin interface. On the left is the database navigation tree with 'departamentos' selected. The main panel displays the 'departamentos' table with 5 rows. A red arrow points to the 'Borrar' button for the first row (codigo 1). The table data is as follows:

codigo	nombre	id_localizacion	id_manager
1	Finanzas	25	10
2	Comercial	70021	92
3	Directivo	5772	58
4	Compras	233	34
5	Logística	10002	150

4. RA09\_e) Se han efectuado borrados y modificaciones sobre la información almacenada.

### ✖ **Borrar**

Se va a recoger el código introducido en el TextField Código que es la Primary Key.

Posteriormente el programa preguntará si está seguro de borrar ese departamento.

Si la respuesta es sí se borrará el registro correspondiente de la tabla departamentos de la BD.

Se rellenará los datos del departamento en los textField.

Por último se mostrarán todos los departamentos que aún quedan en la Tabla.

## Clase GestorBD

```
public Departamento Borrar(int codigo) {
    Departamento depart = new Departamento();
    try {
        //Se llama al metodo consultarCodigoBaseDatos
        depart = consultarCodigoBaseDatos(codigo);
        String resultadoJoptionpanel;
        conectar();

        //Si depart (resultado que obtenemos del método) no es null..
        if (depart != null) {
            //Preguntamos si se desea borrar
            int pregunta = PreguntashowDialog("¿Desea borrar el departamento con código "
                + codigo + "?", option1: "SI", option2: "NO");

            if (pregunta == 0) {

                //Variable que almacena el DELETE
                String sql = "DELETE FROM Departamentos WHERE codigo =?";

                //Preparar el Statement para realizar el delete,
                //indicandole que el placeholder hace referencia al codigo
                preparedStatement = connection.prepareStatement(string: sql);
                preparedStatement.setInt(i: 1, i1: codigo);

                //Ejecuta la sentencia
                preparedStatement.executeUpdate();
                resultadoJoptionpanel = "Departamento eliminado de la BBDD.";
                InfoshowDialog(msg: resultadoJoptionpanel);

            } else {
                resultadoJoptionpanel = "No se ha borrado el Departamento de la BBDD";
                InfoshowDialog(msg: resultadoJoptionpanel);
            }
        }
    }
}
```

- x Se llama al método **consultarCodigoBaseDatos** que será almacenado en el objeto de la clase departamento.

```
//Método para consultar que el registro ya existe en la BD
public Departamento consultarCodigoBaseDatos(int codigo) {

    Departamento depart = new Departamento();
    String resultado;
    try {
        conectar();

        //Variable para consultar dentro de Departamento el codigo introducido
        String sql = "SELECT * FROM Departamentos WHERE codigo = ?";

        //Se prepara el Statement para realizar la consulta, pasamos la variable con la consulta
        //e indicamos único placeholder haciendo referencia al código
        preparedStatement = connection.prepareStatement(string: sql);

        preparedStatement.setInt(i: 1, i1: codigo); //1º placeholder "?" -> codigo = ?

        //Se ejecuta la consulta
        resultSet = preparedStatement.executeQuery();
        //Si el el metodo retorna vacio entonces devuelve que no existe
        if (!resultSet.next()) {
            resultado = "Este registro no existe en la BBDD.";
            InfoshowDialog(msg: resultado);
            return null;
        }

        //Se crea un objeto del resultado
        depart.setCodigo(codigo: resultSet.getInt(i: 1));
        depart.setNombre(nombre: resultSet.getString(i: 2));
        depart.setIdLocalizacion(idLocalisacon: resultSet.getInt(i: 3));
        depart.setIdManager(idManager: resultSet.getInt(i: 4));

        //return el objeto
        return depart;
    } catch (SQLException e) {
        InfoshowDialog("Error al comprobar el departamento: " + e.getMessage());
        return null;
    } finally {
        desconectar();
    }
}
```

## Conectamos con la BD

Preparamos la consulta, indicamos el placeholder y ejecutamos la consulta

Una vez ejecutada realizamos un if que si no existe dicho código en la BD mandar una ventana emergente mostrando que no existe ese registro.

En cambio si la consulta devuelve un registro, almacenamos dicho registro al objeto depart.

Retornamos el depart y cerramos la conexión.



```

public Departamento Borrar(int codigo) {
    Departamento depart = new Departamento();
    try {
        //Se llama al metodo consultarCodigoBaseDatos
        depart = consultarCodigoBaseDatos(codigo);
        String resultadoOptionPane;
        conectar();
    }
}

```

- ✗ Una vez realizada la consulta **conectamos con la BD** ya que anteriormente fue desconectada.

```

//Metodo para conectar con la BD
public void conectar() {
    try {
        //Decir que tipo de bbdd tengo
        Class.forName(className: "com.mysql.jdbc.Driver");
        //Se crea la conexión (url, usuario, pwd)
        connection = DriverManager.getConnection(url:"jdbc:mysql://localhost/15456141a", user: "jcfm", password: "jcfm");
    } catch (ClassNotFoundException | SQLException e) {
        JOptionPane.showMessageDialog(parentComponent:null, "Error de conexión: " + e.getMessage(),
            title: "Error", messageType: JOptionPane.ERROR_MESSAGE);
    }
}

```

**Carga el driver JDBC.**

**Y se crea una conexión en el que se va a indicar donde se encuentra la BD y el login**

```

public Departamento Borrar(int codigo) {
    Departamento depart = new Departamento();
    try {
        //Se llama al metodo consultarCodigoBaseDatos
        depart = consultarCodigoBaseDatos(codigo);
        String resultadoJoptionpanel;
        conectar();

        //Si depart (resultado que obtenemos del método) no es null..
        if (depart != null) {
            //Preguntamos si se desea borrar
            int pregunta = PreguntashowDialog("¿Desea borrar el departamento con código "
                + codigo + "?", option1:"SI", option2:"NO");

            if (pregunta == 0) {

                //Variable que almacena el DELETE
                String sql = "DELETE FROM Departamentos WHERE codigo =?";

                //Preparar el Statement para realizar el delete,
                //indicandole que el placeholder hace referencia al codigo
                preparedStatement = connection.prepareStatement(string: sql);
                preparedStatement.setInt(i: 1, i1: codigo);

                //Ejecuta la sentencia
                preparedStatement.executeUpdate();
                resultadoJoptionpanel = "Departamento eliminado de la BBDD.";
                InfoshowDialog(msg:resultadoJoptionpanel);

            } else {
                resultadoJoptionpanel = "No se ha borrado el Departamento de la BBDD";
                InfoshowDialog(msg:resultadoJoptionpanel);
            }
        }
    }
}

```

- x Si el objeto depart no es null entonces preguntamos si se desea borrar ese departamento que se ha indicado.
- x Si la respuesta de la pregunta es Si entonces entra al siguiente if, guardamos la sentencia del DELETE
- x Preparamos el statement con la variable que almacena la sentencia.
- x Indicamos que el placeholder es el código que se indica (*código que va a ser recogido desde el textField*)
- x Ejecutamos la sentencia del DELETE y mostramos por ventaja emergente que se ha eliminado.
- x Si la respuesta de la pregunta es No entonces se indica mediante una ventaja emergete que no se ha eliminado.
- x Por ultimo se **desconecta de la BD** al entrar en el finally.

```
//Metodo para desconectar con la BD
public void desconectar() {
    try {
        //En caso de que Connection, preparedStatement o ResultSet no sea null se desconecta con la BD
        if (connection != null) {
            connection.close();
        }
        if (preparedStatement != null) {
            preparedStatement.close();
        }
        if (resultSet != null) {
            resultSet.close();
        }
    } catch (SQLException e) {
        InfoshowDialog("Error de desconexión: " + e.getMessage());
    }
}
```

Si connection no es null > cerramos la conexión

Si preparedStatement no es null > cerramos la conexión

Si resultSet no es null > cerramos la conexión

## IFrame Form (Interfaz de usuario)

- ✖ Ahora recogeremos los datos que introduzca el usuario y llamaremos a los diferentes métodos ya creados para realizar las diferentes funciones

```
private void borrarBtActionPerformed(java.awt.event.ActionEvent evt) {
    GestorBD baseDatos = new GestorBD();

    String msg = CamposVaciosJFrame.controlarCampoCodigo(codigo: this.codigoTf.getText());

    if (msg == null) {
        //Llamada al metodo borrar departamento
        Departamento depart = baseDatos.Borrar(codigo: Integer.parseInt(s: this.codigoTf.getText()));

        //Se introduce los campos codigo, nombre, localizacion y manager en sus textfield correspondiente
        this.codigoTf.setText(s: String.valueOf(i: depart.getCodigo()));
        this.nombreTf.setText(s: String.valueOf(i: depart.getNombre()));
        this.idlocalizacionTf.setText(s: String.valueOf(i: depart.getIdLocalizacion()));
        this.idmanagerTf.setText(s: String.valueOf(i: depart.getIdManager()));

        //Llamada del metodo mostrar
        mostrar();
    } else {
        //Si el campo codigo está vacío lo indicamos
        baseDatos.ErrorshowDialog(msg);
    }
}
```

- ✖ Creamos un **objeto de la clase GestorBD**.
- ✖ Llamamos al **método estático controlarCampoCodigo** de la clase CamposVaciosJFrame, para controlar que no se deje el campo código vacío para poder eliminar el departamento.
- ✖ Si devuelve null significa que el campo TextField código no está en blanco entonces **pasamos a llamar al método de borrar** almacenándolo en el objeto de la clase departamento.

- ✖ **Introducimos los campos** código, nombre, id localización y id manager **en los TextField** que serán recogidos del objeto depart anteriormente almacenado del método borrar.
- ✖ Por ultimo llamamos al método mostrar() para mostrar los departamentos en la Tabla con la actualización del borrado.

## Resultado (Interfaz Gráfica)

- ✖ Introducimos un código ya registrado en la BD para realizar el borrado del departamento.

### Datos del Departamento

Código

Nombre

ID Localización

ID Manager

INSERTAR

MODIFICAR

BORRAR

### Mostrar datos del departamento

Código	Nombre	ID Localización	ID Manager
1	Finanzas	25	10
2	Comercial	70021	92
3	Directivo	5772	58
4	Compras	233	34
5	Logística	10002	150

LIMPIAR CAMPOS

MOSTRAR

SALIR

- ✖ Al pulsar en **BORRAR** nos preguntará si se desea borrar el departamento.

### Datos del Departamento

Código

Nombre

ID Localización

ID Manager

INSERTAR

MODIFICAR

BORRAR

### Mostrar datos del departamento

Código	Nombre	ID Localización	ID Manager
1	Finanzas	25	10
2	Comercial	70021	92
3	Directivo	5772	58
4	Compras	233	34
5	Logística	10002	150

LIMPIAR CAMPOS

MOSTRAR

SALIR

?

¿Desea borrar el departamento con código 5?

SI

NO

Al indicar que si entonces **se eliminará el departamento, se auto completará los textFields y se mostrará la modificación en la tabla** en este caso se eliminará el departamento indicado

### Datos del Departamento

Código

5

Nombre

Logística

ID Localización

10002

ID Manager

150

INSERTAR

MODIFICAR

BORRAR

### Mostrar datos del departamento

Código	Nombre	ID Localización	ID Manager
1	Finanzas	25	10
2	Comercial	70021	92
3	Directivo	5772	58
4	Compras	233	34

LIMPIAR CAMPOS

MOSTRAR

SALIR

## Resultado (phpMyAdmin)

phpMyAdmin

Reciente Favoritas

- Nueva
- 15456141a
- Nueva
- departamentos
- bdreacionales
- information\_schema
- mysql
- performance\_schema
- phpmyadmin
- test

Servidor: 127.0.0.1 » Base de datos: 15456141a » Tabla: departamentos

Examinar Estructura SQL Buscar Insertar Exportar

✓ Mostrando filas 0 - 3 (total de 4, La consulta tardó 0,0000 segundos.)

`SELECT * FROM `departamentos``

☐ Perfilando [Editar en línea] [Editar] [Explicar SQL] [Crear código]

☐ Mostrar todo | Número de filas: 25 | Filtrar filas: Buscar en esta tabla

+ Opciones

	codigo	nombre	id_localizacion	id_manager
<input type="checkbox"/> Editar Copiar Borrar	1	Finanzas	25	10
<input type="checkbox"/> Editar Copiar Borrar	2	Comercial	70021	92
<input type="checkbox"/> Editar Copiar Borrar	3	Directivo	5772	58
<input type="checkbox"/> Editar Copiar Borrar	4	Compras	233	34

☐ Seleccionar todo Para los elementos que están marcados: Editar Copiar Exportar

## × *Modificar*

Se va a modificar un departamento introduciendo el código registrado en el textField y la modificación del nombre, id\_localización e id\_manager que se necesite.

Una vez el usuario haya realizado los cambios deseados, al seleccionar la opción de modificar actualizarán los datos en la base de datos.

Se mostrará la modificación rellenando los TextFields *(Si se indica el código y se modifica por ejemplo el nombre entonces el resto de campos recogerá los datos del anterior departamento)*

Se mostrarán todos los departamentos en la Tabla, de modo que se pueda comprobar que efectivamente se ha realizado la modificación.

## Clase GestorBD

```
//Metodo para actualizar Departamentos
public void actualizar(int codigo, String nombre, int id_localizacion, int id_manager) {

    try {
        conectar();
        String resultado;

        //Variable que almacena el UPDATE
        String sql = "UPDATE Departamentos SET nombre = ?, id_localizacion = ?, id_manager = ? WHERE codigo = ?";

        //Se prepara el Statement pasandole la variable que contiene el UPDATE e
        //e indicamos cada placeholder haciendo referencia a la BD que seria:
        preparedStatement = connection.prepareStatement(string: sql);

        preparedStatement.setString(i: 1, string: nombre); //1º placeholder "?" -> nombre = ?
        preparedStatement.setInt(i: 2, i1: id_localizacion); //2º placeholder "?" -> id_localizacion = ?
        preparedStatement.setInt(i: 3, i1: id_manager); //3º placeholder "?" -> id_manager = ?
        preparedStatement.setInt(i: 4, i1: codigo); //4º placeholder "?" -> codigo = ?

        //Se realiza el update
        preparedStatement.executeUpdate();
        resultado = "Actualizado.";
        InfoshowDialog(msg: resultado);

    } catch (SQLException e) {
        InfoshowDialog("Error al borrar departamento: " + e.getMessage());
    } finally {
        desconectar();
    }
}
```

- ✗ Se conectará a la BD con el método `conectar()`;

```
//Metodo para conectar con la BD
public void conectar() {
    try {
        //Decir que tipo de bbdd tengo
        Class.forName(className: "com.mysql.jdbc.Driver");
        //Se crea la conexión (url, usuario, pwd)
        connection = DriverManager.getConnection(url:"jdbc:mysql://localhost/15456141a", user:"jcfm", password:"jcfm");

    } catch (ClassNotFoundException | SQLException e) {
        JOptionPane.showMessageDialog(parentComponent:null, "Error de conexión: " + e.getMessage(),
            title: "Error", messageType:JOptionPane.ERROR_MESSAGE);
    }
}
```

Carga el driver JDBC.

Y se crea una conexión en el que se va a indicar donde se encuentra la BD y el login

```
//Metodo para actualizar Departamentos
public void actualizar(int codigo, String nombre, int id_localizacion, int id_manager) {

    try {
        conectar();
        String resultado;

        //Variable que almacena el UPDATE
        String sql = "UPDATE Departamentos SET nombre = ?, id_localizacion = ?, id_manager = ? WHERE codigo = ?";

        //Se prepara el Statement pasandole la variable que contiene el UPDATE e
        //e indicamos cada placeholder haciendo referencia a la BD que seria:
        preparedStatement = connection.prepareStatement(string: sql);

        preparedStatement.setString(i: 1, string: nombre); //1º placeholder "?" -> nombre = ?
        preparedStatement.setInt(i: 2, i1: id_localizacion); //2º placeholder "?" -> id_localizacion = ?
        preparedStatement.setInt(i: 3, i1: id_manager); //3º placeholder "?" -> id_manager = ?
        preparedStatement.setInt(i: 4, i1: codigo); //4º placeholder "?" -> codigo = ?

        //Se realiza el update
        preparedStatement.executeUpdate();
        resultado = "Actualizado.";
        InfoshowDialog(msg:resultado);

    } catch (SQLException e) {
        InfoshowDialog("Error al borrar departamento: " + e.getMessage());
    } finally {
        desconectar();
    }
}
```

- ✗ Almacenamos en la variable `sql` el **UPDATE** que se va a realizar.
- ✗ Preparamos el **Statement** pasándole la variable `sql`
- ✗ Indicamos cada **placeholder** donde hace referencia al nombre, `id_localizacion`, `id_manager` y el código que es controlado con el **WHERE** de la sentencia.
- ✗ Se ejecuta el **Update** y mostramos mediante una ventana emergente que se ha actualizados
- ✗ Desconectamos la **BD** al entrar en el `finally`

```
//Metodo para desconectar con la BD
public void desconectar() {
    try {
        //En caso de que Connection, preparedStatement o ResultSet no sea null se desconecta con la BD
        if (connection != null) {
            connection.close();
        }
        if (preparedStatement != null) {
            preparedStatement.close();
        }
        if (resultSet != null) {
            resultSet.close();
        }
    } catch (SQLException e) {
        InfoshowDialog("Error de desconexión: " + e.getMessage());
    }
}
```

Si connection no es null > cerramos la conexión

Si preparedStatement no es null > cerramos la conexión

Si resultSet no es null > cerramos la conexión

## Frame form (Interfaz de usuario)

- ✖ Ahora recogeremos los datos que introduzca el usuario y llamaremos a los diferentes métodos ya creados para realizar las diferentes funciones

```
private void modificarActionPerformed(java.awt.event.ActionEvent evt) {
    GestorBD baseDatos = new GestorBD();

    String msg = CamposVaciosJFrame.controlarCampoCodigo(codigo: this.codigoTf.getText());

    if (msg == null) {
        //llamada al metodo ConsultarCodigoBaseDatos
        Departamento depart = baseDatos.consultarCodigoBaseDatos(codigo: Integer.parseInt(s: this.codigoTf.getText()));

        //si nombre es vacio o espacios en blanco = recoge el campo anterior introduciendo este en el textField
        if (this.nombreTf.getText().isEmpty() || this.nombreTf.getText().isBlank()) {
            this.nombreTf.setText(e: depart.getNombre());
        }
        //si id localizacion es vacio o espacios en blanco = recoge el campo anterior
        if (this.idlocalizacionTf.getText().isEmpty() || this.idlocalizacionTf.getText().isBlank()) {
            this.idlocalizacionTf.setText(e: String.valueOf(i: depart.getIdLocalizacion()));
        }
        //si idmanager es vacio o espacios en blanco = recoge el campo anterior
        if (this.idmanagerTf.getText().isEmpty() || this.idmanagerTf.getText().isBlank()) {
            this.idmanagerTf.setText(e: String.valueOf(i: depart.getIdManager()));
        }

        //Una vez comprobada la consulta anterior se llama al metodo Actualizar Departamento
        baseDatos.actualizar(codigo: Integer.parseInt(s: this.codigoTf.getText()), nombre: this.nombreTf.getText(),
            id_localizacion: Integer.parseInt(s: this.idlocalizacionTf.getText()), id_manager: Integer.parseInt(s: this.idmanagerTf.getText()));

        //Llamada del metodo mostrar
        mostrar();
    } else {
        //Si el campo codigo está vacio lo indicamos
        baseDatos.ErrorshowDialog(msg);
    }
}
```



- ✖ Creamos un objeto de la clase GestorBD.
- ✖ Llamamos al método estático `controlarCampoCodigo` de la clase `CamposVaciosJFrame`, para controlar que no se deje el campo código vacío para poder modificar el departamento.
- ✖ Si devuelve `null` significa que el campo `TextField` código no está en blanco
- ✖ Entonces pasamos a llamar al método de `consultarCodigoBaseDatos`, que será almacenado en el objeto `depart` de la clase `departamento`
- ✖ Mediante los `if` se va a controlar que si los `textFields` `nombre`, `id_localizacion` e `id_manager` se encuentra en blanco o con espacios.
- ✖ Se va rellenar los `textFields` con la consulta obtenida con el método `consultarCodigoBaseDatos` y que se han almacenado en el objeto `departamento`.
- ✖ Una vez comprobada la consulta y rellenando los `textFields` en blanco si fuera el caso pasaremos a llamar al método `actualizar`.
- ✖ Por ultimo llamamos al método `mostrar`. Mostrando todos los departamentos actualizados de la BD.

## Resultado (Interfaz Gráfica)

- ✖ Introducimos un código ya registrado en la BD para actualizar ese registro modificando el departamento.
- ✖ Solamente vamos a indicarle un nombre y el `ID_Manager`

The screenshot shows a Java Swing window with two main panels. The left panel, titled 'Datos del Departamento', contains four text input fields: 'Codigo' (with value '1'), 'Nombre' (with value 'Marketing'), 'ID Localización' (empty), and 'ID Manager' (with value '50'). Below these fields are three buttons: 'INSERTAR', 'MODIFICAR', and 'BORRAR'. The right panel, titled 'Mostrar datos del departamento', contains a table with four columns: 'Código', 'Nombre', 'ID Localización', and 'ID Manager'. The table has four rows of data. Below the table are three buttons: 'LIMPIAR CAMPOS', 'MOSTRAR', and 'SALIR'. A red arrow points from the 'Codigo' field in the left panel to the table in the right panel.

Código	Nombre	ID Localización	ID Manager
1	Finanzas	25	10
2	Comercial	70021	92
3	Directivo	5772	58
4	Compras	233	34

- ✖ Al pulsar en **MODIFICAR** el departamento **1** será actualizado a Marketing pero conservando el ID\_Localización anterior.
- ✖ **Se auto completará los TextField** con el departamento modificado recogiendo los datos faltantes y se mostrará la modificación en la Tabla.

**Datos del Departamento**

Código: 1

Nombre: Marketing

ID Localización: 25

ID Manager: 50

INSERTAR MODIFICAR BORRAR

**Mostrar datos del departamento**

Código	Nombre	ID Localización	ID Manager
1	Marketing	25	50
2	Comercial	70021	92
3	Directivo	5772	58
4	Compras	233	34

LIMPIAR CAMPOS MOSTRAR SALIR

## Resultado (phpMyAdmin)

phpMyAdmin

Reciente Favoritas

Nueva 15456141a

Nueva departamentos

bdreacionales

information\_schema

mysql

performance\_schema

phpmyadmin

test

Servidor: 127.0.0.1 » Base de datos: 15456141a » Tabla: departamentos

Examinar Estructura SQL Buscar Insertar Exportar

Mostrando filas 0 - 3 (total de 4, La consulta tardó 0,0000 segundos.)

SELECT \* FROM `departamentos`

Perfilando [Editar en línea] [Editar] [Explicar SQL] [Crear código F]

Mostrar todo | Número de filas: 25 | Filtrar filas: Buscar en esta tabla

+ Opciones

	codigo	nombre	id_localizacion	id_manager
<input type="checkbox"/> Editar Copiar Borrar	1	Marketing	25	50
<input type="checkbox"/> Editar Copiar Borrar	2	Comercial	70021	92
<input type="checkbox"/> Editar Copiar Borrar	3	Directivo	5772	58
<input type="checkbox"/> Editar Copiar Borrar	4	Compras	233	34

Seleccionar todo Para los elementos que están marcados: Editar Copiar Exportar

## 5. RA09\_e) Se han efectuado borrados y modificaciones sobre la información almacenada.

### × *Mostrar*

Para la función mostrar se va a realizar una consulta sobre la tabla departamentos de la base de datos y mostrar toda la información en la Tabla.

### Clase GestorBD

```
//Metodo para consultar la BBDD y devolver un arrayList con el resultado de la consulta
public ArrayList<Departamento> mostrar() {
    //Se crea un arrayList
    ArrayList<Departamento> arrayDeparts = new ArrayList<Departamento>();

    try {
        conectar();
        //Variable que almacena el SELECT
        String sql = "Select * from departamentos";

        //Se prepara la consulta y se ejecuta almacenando esta consulta completa en resultSet
        Statement consulta = connection.createStatement();
        resultSet = consulta.executeQuery(string: sql);

        //Bucle en el que se recorre linea por linea el resultSet almacenando
        //este en el arrayList arrayDeparts
        while (resultSet.next()) {
            //Se crea objeto Departamento
            Departamento depart = new Departamento();

            //Se va añadiendo cada campo de la BD en el objeto depart con sus setters
            depart.setCodigo(codigo: resultSet.getInt(i: 1));
            depart.setNombre(nombre: resultSet.getString(i: 2));
            depart.setIdLocalizacion(idLocalizacon: resultSet.getInt(i: 3));
            depart.setIdManager(idManager: resultSet.getInt(i: 4));

            //Se añade el objeto al arrayList llamado arrayDeparts
            arrayDeparts.add(e: depart);
        }
        return arrayDeparts;
    } catch (SQLException e) {
        InfoshowDialog("Error al mostrar departamentos: " + e.getMessage());
        return null;
    } finally {
        desconectar();
    }
}
```

- ✖ Se **crea un arrayList** objeto de la clase Departamento
- ✖ Conectamos con la BD con el método **conectar();**

```
//Metodo para conectar con la BD
public void conectar() {
    try {
        //Decir que tipo de bbdd tengo
        Class.forName(className: "com.mysql.jdbc.Driver");
        //Se crea la conexión (url, usuario, pwd)
        connection = DriverManager.getConnection(url:"jdbc:mysql://localhost/15456141a", user:"jcfm", password:"jcfm");

    } catch (ClassNotFoundException | SQLException e) {
        JOptionPane.showMessageDialog(parentComponent:null, "Error de conexión: " + e.getMessage(),
            title: "Error", messageType:JOptionPane.ERROR_MESSAGE);
    }
}
```

**Carga el driver JDBC.**

**Y se crea una conexión en el que se va a indicar donde se encuentra la BD y el login**

- ✖ Una vez conectada la BD se almacenará en la variable **sql** la consulta SELECT de la tabla departamentos.
- ✖ Se prepara la consulta creando el **Statement** y realizamos la consulta con el método **executeQuery()** pasandole la variable **sql** por parámetros.
- ✖ Entraremos en el **bucle while** se **recorre linea por linea el resultSet para almacenar este en el arrayList** arrayDeparts.
- ✖ En cada vuelta del bucle **se añade cada campo de la BD en el objeto depart** y **se añade el objeto en el arrayDeparts.**
- ✖ Entrará al finally que desconectará la BD

```
//Metodo para desconectar con la BD
public void desconectar() {
    try {
        //En caso de que Connection, preparedStatement o ResultSet no sea null se desconecta con la BD
        if (connection != null) {
            connection.close();
        }
        if (preparedStatement != null) {
            preparedStatement.close();
        }
        if (resultSet != null) {
            resultSet.close();
        }
    }
}
```

**Si connection no es null > cerramos la conexión**

**Si preparedStatement no es null > cerramos la conexión**

**Si resultSet no es null > cerramos la conexión**

## JFrame form (Interfaz de usuario)

En el JFrame Form se ha creado un método mostrar. Este método va a llamar al otro método mostrar que será almacenado en un arraylist.

```
private void mostrar() {  
    //Se crea un objeto dm de la clase DefaultTableModel Para establecer el modelo a la  
    //TablaDepartamento y se reinicia los elementos de la lista (borrarlos en cada llamada del metodo mostrar)  
    DefaultTableModel dm = (DefaultTableModel) tablaDepartamento.getModel();  
    dm.getDataVector().removeAllElements();  
  
    //Objetos de la clase conector base de datos  
    GestorBD baseDatos = new GestorBD();  
  
    //ArrayList para almacenar cada departamento  
    ArrayList<Departamento> arrayDeparts = baseDatos.mostrar();  
  
    //Bucle para recorrer el arraylist.  
    for (int i = 0; i < arrayDeparts.size(); i++) {  
        //Se recogen los datos mediante los getters y se van almacenando en el array datosTablaDepartamento  
        String[] datosTablaDepartamento = {  
            Integer.toString(i: arrayDeparts.get(index: i).getCodigo()),  
            arrayDeparts.get(index: i).getNombre(),  
            Integer.toString(i: arrayDeparts.get(index: i).getIdLocalizacion()),  
            Integer.toString(i: arrayDeparts.get(index: i).getIdManager())  
        };  
  
        //Se añade cada objeto en una linea nueva del array datosTablaDepartamento  
        dm.addRow(rowData:datosTablaDepartamento);  
    }  
}
```

- x Se crea un objeto de la clase DefaultTableModel con esto establecemos el modelo en tablaDepartamento
- x Reiniciamos el dm (objeto de la clase DefaultTableModel) para que cada vez que se ejecute este método la tabla se reinicie a 0 y no se dupliquen los campos.
- x Creamos un objeto de la clase baseDatos para poder llamar al método mostrar de la misma y posteriormente llamamos a ese método almacenándolo en un arrayList.
- x Recorremos el arrayList mediante el bucle for. Este arrayList será almacenado en un array llamado datosTablaDepartamento y en cada vuelta del bucle se irán recogiendo los departamentos.
- x Por último añadimos el array que se ha obtenido en el bucle for en el dm.

```
private void mostrarBtActionPerformed(java.awt.event.ActionEvent evt) {  
    //Llamada del metodo mostrar  
    mostrar();  
}
```

- x En el botón mostrar se llamó al método mostrar() que se encuentra en el propio JFrame Form.

## Resultado (Interfaz Gráfica)

The interface is divided into two main sections. The left section, titled "Datos del Departamento", contains four input fields labeled "Código", "Nombre", "ID Localización", and "ID Manager". Below these fields are three buttons: "INSERTAR", "MODIFICAR", and "BORRAR". The right section, titled "Mostrar datos del departamento", features a table with four columns: "Código", "Nombre", "ID Localización", and "ID Manager". The table body is currently empty. Below the table are three buttons: "LIMPIAR CAMPOS", "MOSTRAR", and "SALIR".

- ✖ Pulsamos en el botón **MOSTRAR** y obtendremos la lista de departamentos que hay en la BD

The interface is the same as the previous screenshot, but the table in the "Mostrar datos del departamento" section now contains data. A red arrow points to the "MOSTRAR" button, indicating the action that triggered the data display.

Código	Nombre	ID Localización	ID Manager
1	Marketing	25	50
2	Comercial	70021	92
3	Directivo	5772	58
4	Compras	233	34

## ✖ *Limpiar Campos y salir*

Se han creado dos botones más:

### 1 - Función de limpiar los campos textField:

```
private void limpiarCampos() {  
    this.codigoTf.setText("");  
    this.nombreTf.setText("");  
    this.idlocalizacionTf.setText("");  
    this.idmanagerTf.setText("");  
}
```

```
private void limpiarCamposBtActionPerformed(java.awt.event.A  
    //Limpiamos los campos TextField  
    limpiarCampos();  
}
```

### 2 - Función para salir de la aplicación:

```
private void salirBtActionPerformed(java.  
    System.exit(status: 0);  
}
```