
# CLASE TEÓRICA 6 / 12

## ÍNDICE DE CONTENIDO

## 6.1 Introducción a las Bases de Datos

Una Base de Datos es una entidad en la cual se pueden almacenar datos de manera estructurada, con la menor redundancia posible. Diferentes programas y diferentes usuarios deben poder utilizar estos datos. Por lo tanto, el concepto de Base de Datos generalmente está relacionado con el de red ya que se debe poder compartir esta información. "Sistema de información" es el término general utilizado para la estructura global que incluye todos los mecanismos para compartir datos que se han instalado.

Una Base de Datos proporciona a los usuarios el acceso a datos, que pueden visualizar, ingresar o actualizar, en concordancia con los derechos de acceso que se les hayan otorgado. Tiende a ser más útil a medida que la cantidad de datos almacenados crece.

Una Base de Datos puede ser local, es decir que puede utilizarla sólo un usuario en un equipo, o puede ser distribuida, es decir que la información se almacena en equipos remotos y se accede a ella a través de una red.

La principal ventaja de utilizar bases de datos es que múltiples usuarios pueden acceder al mismo tiempo.

### 6.1.1 Administración de Base de Datos

En la era digital rápidamente surgió la necesidad de contar con un sistema de administración para controlar tanto los datos como los usuarios. La administración de bases de datos se realiza con un sistema llamado **DBMS** (Database Management System). El DBMS es un conjunto de servicios (aplicaciones de software) para administrar bases de datos, que permite:

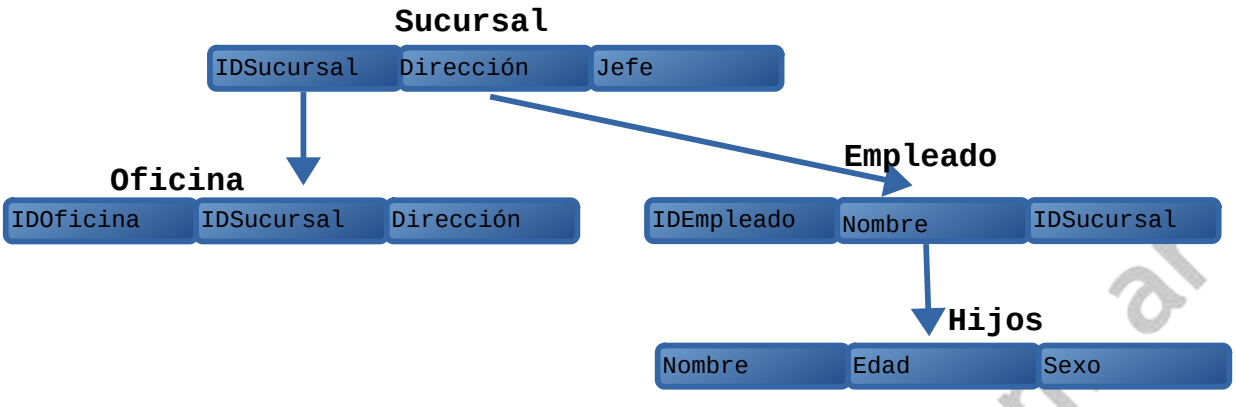
- un fácil acceso a los datos
- el acceso a la información por parte de múltiples usuarios
- la manipulación de los datos encontrados en la Base de Datos (insertar, eliminar, editar, buscar) .

En español se indica como Sistema Gestor de Base de Datos **SGBD**.

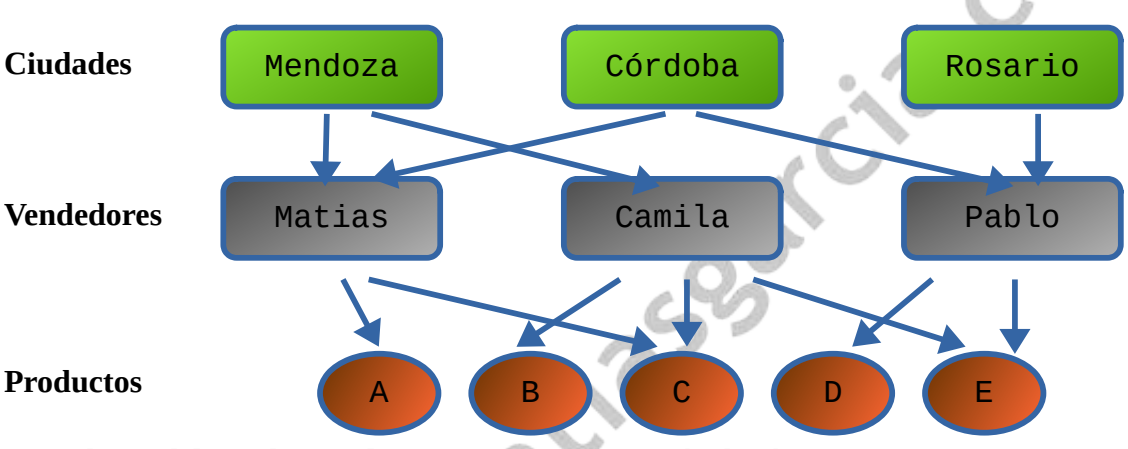
### 6.1.2 Diferentes modelos de Base de Datos

Las bases de datos aparecieron a finales de la década de 1960, cuando surgió la necesidad de contar con un sistema de administración de información flexible. Algunos de los modelos de DBMS, que se distinguen según cómo representan los datos almacenados son:

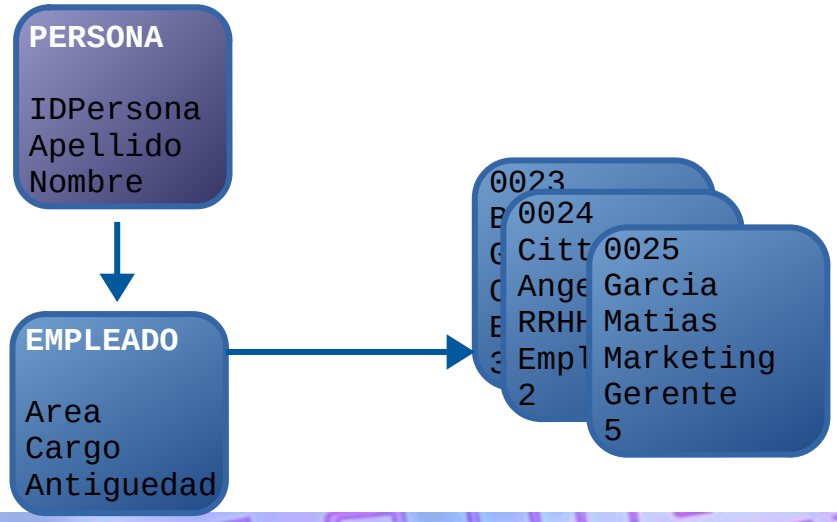
- **El modelo jerárquico**: los datos se organizan jerárquicamente mediante un árbol invertido. Este modelo utiliza punteros para navegar por los datos almacenados. Fue el primer modelo DBMS.




- **El modelo de red**: al igual que el modelo jerárquico, este modelo utiliza punteros hacia los datos almacenados. Sin embargo, no necesariamente utiliza una estructura de árbol invertido.




- **El modelo relacional** (**RDBMS**, *Relational database management system [Sistema de administración de bases de datos relacionales]*): los datos se almacenan en tablas de dos dimensiones (filas y columnas). Los datos se manipulan según la teoría relacional de matemáticas.
- **El modelo deductivo**: los datos se representan como una tabla, pero se manipulan mediante cálculos de predicados. Es un Sistema que permite derivar nuevas informaciones a partir de las introducidas explícitamente en la Base por el usuario.
- **El modelo de orientación a objetos** (**ODBMS**, *object-oriented database management system [sistema de administración de bases de datos orientadas a objetos]*): los datos se almacenan como objetos, que son estructuras denominadas *clases* que muestran los datos que contienen.



### 6.1.3 Bases de Datos Relacionales

Hoy en día hay que tener en cuenta que la inmensa mayoría de los DBMS administran bases de datos relacionales. Éstas son bases de datos que permiten organizar los datos en tablas que después se relacionan mediante campos clave y que trabajan con el lenguaje estándar conocido como SQL.

Cada tabla es una serie de filas y columnas, en la que cada fila es un registro y cada columna un campo. Cada campo representa un dato de los elementos almacenados en la tabla (nombre, DNI,...). Cada registro representa un elemento de la tabla (la señora Eva Jiménez , el señor Matias García,...). No puede aparecer dos veces el mismo registro, por lo que uno o más campos forman lo que se conoce como clave principal.

La clave principal no se puede repetir en dos registros y permite que los datos se relacionen.

SQL es el lenguaje standard internacional que se utiliza para consulta y manipulación de los datos en una Base de Datos relacional.

Algunos de los sistemas de administración de bases de datos relacionales RDBMS que hay en la actualidad son:

- Oracle Database
- IBM DB2
- Microsoft SQL Server
- MySQL
- PostgreSQL
- MongoDB
- MariaDB
- Borland Paradox
- Sybase
- SQLite

JAVA incluye en su JDK un RDBMS puro de JAVA, llamado JAVA DB.

MySQL y su fork MariaDB son los sistemas de administración de bases de datos de código fuente abierto más populares en la actualidad. Todos los ejemplos en este tutorial son utilizando MySQL o MariaDB.

Los programas en JAVA se comunican con las bases de datos y manipulan sus datos utilizando la API JDBC (JAVA Data Base Connectivity), es una biblioteca de clases que permite la conexión con Bases de Datos que soporten SQL utilizando JAVA, con lo que se logra que la aplicación sea independiente de la plataforma y que se pueda mover de un sistema gestor de bases de datos a otro (por ej de Oracle a MySQL o a Microsoft SQL Server). Un controlador (driver) de JDBC permite a las aplicaciones de JAVA conectarse a una Base de Datos de un DBMS determinado y manipular la BD mediante la API JDBC.

La mayoría de los sistemas de administración de bases de datos populares incluyen controladores (drivers) para JDBC. En este tutorial emplearemos la tecnología JDBC para manipular datos de MySQL/ MariaDB. Estas técnicas se pueden utilizar con otras bases de datos relacionales siempre y cuando éstas BD tengan un controlador de JDBC.

## 6.2 Creación de una Base de Datos

Para crear una Base de Datos relacional desde el Administrador de Base de Datos MySql o MariaDB sería conveniente instalar alguna utilidad con una interfaz web gráfica como por ejemplo phpMyAdmin.

phpMyAdmin es una aplicación web que nos sirve para interactuar con una Base de Datos de forma muy sencilla y desde una interfaz web. Podemos crear bases de datos, tablas, borrar o modificar datos, añadir registros, hacer copias de seguridad, etc. Es una aplicación tan útil que casi todos los hosting disponen de ella. Además, la utilizaremos para crear los usuarios para así poder acceder a las bases de datos de forma segura. Al ser una aplicación web escrita en PHP, necesita del servidor web Apache y el motor MySQL o MariaDB para poder funcionar.

Esto implica que tenemos que instalar todo lo anterior y configurarlo.

Para que esa tarea sea más fácil existen también otras utilidades web que configuran todo automáticamente como ser

- XAMPP es un servidor independiente de la plataforma, Software Libre, que consiste principalmente en la Base de Datos MariaDB, el Servidor Web Apache y los intérpretes para lenguajes de script: PHP y Perl.
- WAMP de MS Windows es un entorno de desarrollo Web. Permite crear aplicaciones web y ejecutarlas con Apache, PHP y la Base de Datos MySQL. Se puede utilizar PHPMyAdmin para administrar más fácilmente las bases de datos.
- EASYPHP Programa que instala en un solo paso el servidor Apache, junto con el módulo para programación en PHP y la Base de Datos MySQL.

Luego de haber instalado XAMPP o WAMP se puede acceder a la herramienta phpMyAdmin, que abre una página web desde la cual podemos crear la Base de Datos con sus tablas, agregar registros a las tablas, crear cuentas de usuarios con sus privilegios, para el acceso a la Base de Datos, etc.

**Ver Anexo 1.**

## 6.3 Conexión y consulta a una Base de Datos desde JAVA

Antes de poder conectarse a una Base de Datos desde un programa escrito en JAVA, debemos tener cargado el conector JDBC correspondiente.

Si no trabajamos con Maven se deberá instalar en la PC el driver correspondiente para poder utilizar JDBC que es el connector/j que bajamos de Internet.

Maria DB: https://downloads.mariadb.org/connector-java/

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

Una vez descargado, se descromprime el archivo o se ejecuta el instalador del driver creando una carpeta de la cual debemos copiar el archivo mysql-connector-java-VERSION-bin.jar o mariadb-java-client-VERSION.jar en la subcarpeta \jre\lib\ext de donde tengamos instalado el JDK y JRE de JAVA.

Si trabajamos con Maven simplemente habrá que agregar la dependencia correspondiente en el POM.

Para MariaDB:

```
<!-- https://mvnrepository.com/artifact/org.mariadb.jdbc/mariadb-java-client -->
<dependency>
    <groupId>org.mariadb.jdbc</groupId>
    <artifactId>mariadb-java-client</artifactId>
    <version>2.7.2</version>
</dependency>
```

Para MySQL:

```
<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.23</version>
</dependency>
```

Para conectarnos y hacer consultas a una Base de Datos debemos definir 3 objetos:

- `Connection conexion // maneja la conexión`
- `Statement instruccion // instrucción de consulta`
- `ResultSet conjuntoResultados // maneja los resultados`

y luego ejecutar las siguientes tareas:

1. **Cargar el controlador o driver**

```
Class.forName(CONTROLADOR);
```

Donde ***CONTROLADOR*** es una variable donde se define el controlador correspondiente a la base de datos que vayamos a utilizar. Ej MySQL → `String CONTROLADOR = "com.mysql.cj.jdbc.Driver";`

| RDBMS | Driver |
|---|---|
| MySQL | com.mysql.cj.jdbc.Driver |
| MariaDB | org.mariadb.jdbc.Driver |
| PostgreSQL | org.postgresql.Driver |
| ORACLE | oracle.jdbc.driver.OracleDriver |
| DB2 | COM.ibm.db2.jdbc.app.DB2Driver |
| MS SQL Server | com.microsoft.sqlserver.jdbc.SQLServerDrive |
| Sybase | ncom.sybase.jdbc2.jdbc.SybDriver |
| Java DB / Apache Derby | org.apache.derby.jdbc.EmbeddedDriver |

2. **Establecer la conexión a la Base de Datos**

```
conexion = DriverManager.getConnection(URL_BASEDATOS, "usuario", "contraseña");
```

Donde ***URL_BASEDATOS*** es una variable donde se define la URL donde encontrar la base de datos.

Debe indicarse el `usuario` y `contraseña` de la misma para tener acceso.

| RDBMS | Formato URL de acceso a Base de Datos |
|---|---|
| MySQL | `jdbc:mysql://nombrehost:numeroPuerto/nombreBaseDatos` |
| MariaDB | `jdbc:mariadb://nombrehost:numeroPuerto/nombreBaseDatos` |
| PostgreSQL | `jdbc:postgresql://nombrehost:numeroPuerto/nombreBaseDatos` |
| ORACLE | `jdbc:oracle:thin:@nombrehost:numeroPuerto:nombreBaseDatos` |
| DB2 | `jdbc:db2:nombrehost:numeroPuerto/nombreBaseDatos` |
| MS SQL Server | `jdbc:sqlserver://`<br>`nombrehost.numeroPuerto;nombreBaseDatos=nombreBaseDatos` |
| Sybase | `jdbc:sybase:Tds:nombrehost:numeroPuerto/nombreBaseDatos` |
| Java DB / Apache Derby | `jdbc:derby:nombreBaseDatos (incrustado)`<br>`jdbc:derby://nombrehost:numeroPuerto/nombreBaseDatos (red)` |

### 3. Crear un objeto `Statement` para consultar la Base de Datos

```
instruccion = conexion.createStatement();
```

### 4. Efectuar la consulta a la Base de Datos

```
conjuntoResultados = instruccion.executeQuery("SELECT * FROM profesores");

nro_filas = instruccion.executeUpdate("UPDATE clientes SET sexo='M' WHERE id=5");
```

### 5. Mostrar los resultados de la consulta

Ej.

```
import java.sql.Connection;
import java.sql.Statement;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;

public class MostrarProfesores {
    // nombre del controlador de JDBC y URL de la base de datos
    static final String CONTROLADOR = "com.mysql.cj.jdbc.Driver";
    static final String URL_BASEDATOS = "jdbc:mysql://localhost/instituto";

    // inicia la aplicación
    public static void main(String args[]) {
          Connection conexion = null; // maneja la conexión
          Statement instruccion = null; // instrucción de consulta
          ResultSet conjuntoResultados = null; // maneja los resultados
          // se conecta a la base de datos libros y realiza una consulta
          try {
                // carga la clase controlador
                Class.forName(CONTROLADOR);
                // establece la conexión a la base de datos
```

```
                conexion = DriverManager.getConnection(URL_BASEDATOS, "root", "");
//usuario por defecto sin pass
                // crea objeto Statement para consultar la base de datos
                instruccion = conexion.createStatement();
                // consulta la base de datos
                conjuntoResultados = instruccion.executeQuery("SELECT IDProfesor,
Apellido, Nombre FROM profesores");
                // procesa los resultados de la consulta
                ResultSetMetaData metaDatos = conjuntoResultados.getMetaData();
                int numeroDeColumnas = metaDatos.getColumnCount();
                System.out.println("Tabla Profesores de la base de datos
Instituto:\n");
                for (int i = 1; i <= numeroDeColumnas; i++)
                      System.out.printf("%-8s\t", metaDatos.getColumnName(i));
                System.out.println();
                while (conjuntoResultados.next()) {
                      for (int i = 1; i <= numeroDeColumnas; i++)
                            System.out.printf("%-8s\t",
conjuntoResultados.getObject(i));
                      System.out.println();
                } // fin de while
          } // fin de try
          catch (SQLException excepcionSql) {
                excepcionSql.printStackTrace();
          } // fin de catch
          catch (ClassNotFoundException noEncontroClase) {
                noEncontroClase.printStackTrace();
          } // fin de catch
          finally // asegura que conjuntoResultados, instruccion y conexion estén
                // cerrados
          {
                try {
                      conjuntoResultados.close();
                      instruccion.close();
                      conexion.close();
                } // fin de try
                catch (Exception excepcion) {
                      excepcion.printStackTrace();
                }
          }
    }
}
```

Si la Base de Datos estuviese en otro servidor, que no es el local, se reemplazaría la palabra **`localhost`** por el nro de IP de la computadora que aloja la Base de Datos, o su URL.

## 6.4 Aplicación Swing para consultar una Base de Datos

El código que sigue corresponde a la clase **ventana** que contiene el **JTable** que mostrará la información de la consulta que se haga en ejecución.

En el MVC (Modelo Vista Controlador) la clase **VentanaResultadosConsulta** representa la Vista.

```
// Cuando se abre la aplicación se muestran los datos de una
// consulta predeterminada con los datos
// de la tabla Profesores de la Base de Datos instituto
// luego si se cambia la consulta muestra los resultados de la nueva
// consulta

import java.awt.BorderLayout;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.sql.SQLException;
import javax.swing.JFrame;
import javax.swing.JTextArea;
import javax.swing.JScrollPane;
import javax.swing.ScrollPaneConstants;
import javax.swing.JTable;
import javax.swing.JOptionPane;
import javax.swing.JButton;
import javax.swing.Box;

public class VentanaResultadosConsulta extends JFrame
{
   // URL de la Base de Datos, nombre de usuario y contraseña para JDBC
   static final String CONTROLADOR = "com.mysql.cj.jdbc.Driver";
   static final String URL_BASEDATOS = "jdbc:mysql://localhost/instituto";
   static final String USER = "root";
   static final String PASS = "";

   // la consulta predeterminada obtiene todos los datos de la tabla profesores
   static final String CONSULTA_PREDETERMINADA = "SELECT * FROM profesores";

   private ResultSetTableModel modeloTabla;
   private JTextArea areaConsulta;

   // crea objeto ResultSetTableModel y GUI
   public VentanaResultadosConsulta()
   {
      super( "Visualización de los resultados de la consulta" );

      // crea objeto ResultSetTableModel
      // y muestra la consulta de BD en un JTable
      try
      {
         // crea objeto TableModel para los resultados de la consulta SELECT *
FROM autores
```

```
         modeloTabla = new ResultSetTableModel( CONTROLADOR, URL_BASEDATOS, USER,
PASS, CONSULTA_PREDETERMINADA );

         // establece objeto JTextArea en el que el usuario escribe las consultas
         areaConsulta = new JTextArea( CONSULTA_PREDETERMINADA, 3, 100 );
         areaConsulta.setWrapStyleWord( true );
         areaConsulta.setLineWrap( true );

         JScrollPane scrollPane = new JScrollPane( areaConsulta,
            ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED,
            ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER );

         // establece objeto JButton para enviar las consultas
         JButton botonEnviar = new JButton( "Enviar consulta" );

         // crea objeto Box para manejar la colocación de areaConsulta y
         // botonEnviar en la GUI
         Box boxNorte = Box.createHorizontalBox();
         boxNorte.add( scrollPane );
         boxNorte.add( botonEnviar );

         // crea delegado de JTable para modeloTabla
         JTable tablaResultados = new JTable( modeloTabla );

         // coloca los componentes de la GUI en el panel de contenido
         add( boxNorte, BorderLayout.NORTH );
         add( new JScrollPane( tablaResultados ), BorderLayout.CENTER );

         // crea componente de escucha de eventos para botonEnviar
         botonEnviar.addActionListener(

            new ActionListener()
            {
               // pasa la consulta al modelo de la tabla
               public void actionPerformed( ActionEvent evento )
               {
                  // realiza una nueva consulta
                  try
                  {
                     modeloTabla.establecerConsulta( areaConsulta.getText() );
                  } // fin de try
                  catch ( SQLException excepcionSql )
                  {
                     JOptionPane.showMessageDialog( null,
                        excepcionSql.getMessage(), "Error en Base de Datos",
                        JOptionPane.ERROR_MESSAGE );

                     // trata de recuperarse de una consulta inválida del usuario
                     // ejecutando la consulta predeterminada
                     try
                     {
                        modeloTabla.establecerConsulta(
CONSULTA_PREDETERMINADA );
```

```
                        areaConsulta.setText( CONSULTA_PREDETERMINADA );
                     } // fin de try
                     catch ( SQLException excepcionSql2 )
                     {
                        JOptionPane.showMessageDialog( null,
                           excepcionSql2.getMessage(), "Error en Base de Datos",
                           JOptionPane.ERROR_MESSAGE );

                        // verifica que esté cerrada la conexión a la Base de Datos
                        modeloTabla.desconectarDeBaseDatos();

                        System.exit( 1 ); // termina la aplicación
                     } // fin de catch interior
                  } // fin de catch exterior
               } // fin de actionPerformed
            }  // fin de la clase interna ActionListener
         ); // fin de la llamada a addActionListener


         setSize( 500, 250 ); // establece el tamaño de la ventana
         setVisible( true ); // muestra la ventana

      } // fin de try
      catch ( ClassNotFoundException noEncontroClase )
      {
         JOptionPane.showMessageDialog( null,
            "No se encontro controlador de Base de Datos", "No se encontró el
controlador",
            JOptionPane.ERROR_MESSAGE );

         System.exit( 1 ); // termina la aplicación
      } // fin de catch
      catch ( SQLException excepcionSql )
      {
         JOptionPane.showMessageDialog( null, excepcionSql.getMessage(),
            "Error en Base de Datos", JOptionPane.ERROR_MESSAGE );

         // verifica que esté cerrada la conexión a la Base de Datos
         modeloTabla.desconectarDeBaseDatos();

         System.exit( 1 ); // termina la aplicación
      } // fin de catch

     // cierra la ventana cuando el usuario sale de la aplicación (se sobrescribe
      // el valor predeterminado de HIDE_ON_CLOSE)
      setDefaultCloseOperation( DISPOSE_ON_CLOSE );

 // verifica que esté cerrada la conexión a la Base de Datos cuando el usuario sale
de la aplicación
      addWindowListener(

         new WindowAdapter()
         {
```

```
        // se desconecta de la Base de Datos y sale cuando se ha cerrado la ventana
            public void windowClosed( WindowEvent evento )
            {
               modeloTabla.desconectarDeBaseDatos();
               System.exit( 0 );
            } // fin del método windowClosed
         } // fin de la clase interna WindowAdapter
      ); // fin de la llamada a addWindowListener
   } // fin del constructor de VentanaResultadosConsulta

   // ejecuta la aplicación

} // fin de la clase VentanaResultadosConsulta
```

A continuación se muestra el código de la clase que deriva de la clase **`AbstractTableModel`** que se asociará al **`JTable`** para mostrar los datos de la consulta

En el MVC (Modelo Vista Controlador) la clase **`ResultSetTableModel`** representa el Controlador.

```
// Un objeto TableModel que suministra datos ResultSet a un objeto JTable.
import java.sql.Connection;
import java.sql.Statement;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import javax.swing.table.AbstractTableModel;

// las filas y columnas del objeto ResultSet se cuentan desde 1 y
// las filas y columnas del objeto JTable se cuentan desde 0. Al procesar
// filas o columnas de ResultSet para usarlas en un objeto JTable, es
// necesario sumar 1 al número de fila o columna para manipular
// la columna apropiada del objeto ResultSet (es decir, la columna 0 de JTable
// es la columna 1 de ResultSet y la fila 0 de JTable es la fila 1 de ResultSet).
public class ResultSetTableModel extends AbstractTableModel {
    private Connection conexion;
    private Statement instruccion;
    private ResultSet conjuntoResultados;
    private ResultSetMetaData metaDatos;
    private int numeroDeFilas;

    // lleva la cuenta del estado de la conexión a la Base de Datos
    private boolean conectadoABaseDatos = false;

    // el constructor inicializa conjuntoResultados y obtiene su objeto de
    // metadatos y determina el número de filas

    public ResultSetTableModel(String controlador, String url, String
nombreusuario, String contrasenia,
                String consulta) throws SQLException, ClassNotFoundException {
          // se conecta a la Base de Datos
          Class.forName(controlador);
```

```
            conexion = DriverManager.getConnection(url, nombreusuario, contrasenia);

            // crea objeto Statement para consultar la Base de Datos
            instruccion =
conexion.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_READ_ONLY);

            // actualiza el estado de la conexión a la Base de Datos
            conectadoABaseDatos = true;

            // establece consulta y la ejecuta
            establecerConsulta(consulta);
      } // fin del constructor ResultSetTableModel

      // obtiene la clase que representa el tipo de la columna
      public Class getColumnClass(int columna) throws IllegalStateException {
            // verifica que esté disponible la conexión a la Base de Datos
            if (!conectadoABaseDatos)
                  throw new IllegalStateException("No hay conexion a la Base de
Datos");

            // determina la clase de JAVA de la columna
            try {
                  String nombreClase = metaDatos.getColumnClassName(columna + 1);

                  // devuelve objeto Class que representa a nombreClase
                  return Class.forName(nombreClase);
            } // fin de try
            catch (Exception excepcion) {
                  excepcion.printStackTrace();
            } // fin de catch

            return Object.class; // si ocurren problemas en el código anterior,
                                                // asume el tipo Object
      } // fin del método getColumnClass

      // obtiene el número de columnas en el objeto ResultSet
      public int getColumnCount() throws IllegalStateException {
            // verifica que esté disponible la conexión a la Base de Datos
            if (!conectadoABaseDatos)
                  throw new IllegalStateException("No hay conexión a la Base de
Datos");

            // determina el número de columnas
            try {
                  return metaDatos.getColumnCount();
            } // fin de try
            catch (SQLException excepcionSql) {
                  excepcionSql.printStackTrace();
            } // fin de catch

            return 0; // si ocurren problemas en el código anterior, devuelve 0 para
                              // el número de columnas
```

```
    } // fin del método getColumnCount

    // obtiene el nombre de una columna específica en el objeto ResultSet
    public String getColumnName(int columna) throws IllegalStateException {
          // verifica que esté disponible la conexión a la Base de Datos
          if (!conectadoABaseDatos)
                throw new IllegalStateException("No hay conexion a la Base de
Datos");

          // determina el nombre de la columna
          try {
                return metaDatos.getColumnName(columna + 1);
          } // fin de try
          catch (SQLException excepcionSql) {
                excepcionSql.printStackTrace();
          } // end catch

          return ""; // si hay problemas, devuelve la cadena vacía para el nombre
                           // de la columna
    } // fin del método getColumnName

    // devuelve el número de filas en el objeto ResultSet
    public int getRowCount() throws IllegalStateException {
          // verifica que esté disponible la conexión a la Base de Datos
          if (!conectadoABaseDatos)
                throw new IllegalStateException("No hay conexion a la Base de
Datos");

          return numeroDeFilas;
    } // fin del método getRowCount

    // obtiene el valor en la fila y columna específicas
    public Object getValueAt(int fila, int columna) throws IllegalStateException {
          // verifica que esté disponible la conexión a la Base de Datos
          if (!conectadoABaseDatos)
                throw new IllegalStateException("No hay conexion a la Base de
Datos");

          // obtiene un valor en una fila y columna especificadas del objeto
          // ResultSet
          try {
                conjuntoResultados.absolute(fila + 1);
                return conjuntoResultados.getObject(columna + 1);
          } // fin de try
          catch (SQLException excepcionSql) {
                excepcionSql.printStackTrace();
          } // fin de catch

          return ""; // si hay problemas, devuelve el objeto cadena vacía
    } // fin del método getValueAt

    // establece nueva cadena de consulta en la Base de Datos
    public void establecerConsulta(String consulta) throws SQLException,
```

```
IllegalStateException {
          // verifica que esté disponible la conexión a la Base de Datos
          if (!conectadoABaseDatos)
                throw new IllegalStateException("No hay conexión a la Base de
Datos");

          // especifica la consulta y la ejecuta
          conjuntoResultados = instruccion.executeQuery(consulta);

          // obtiene metadatos para el objeto ResultSet
          metaDatos = conjuntoResultados.getMetaData();

          // determina el número de filas en el objeto ResultSet
          conjuntoResultados.last(); // avanza a la última fila
          numeroDeFilas = conjuntoResultados.getRow(); // obtiene el número de

// fila

          // notifica al objeto JTable que el modelo ha cambiado
          fireTableStructureChanged();
    } // fin del método establecerConsulta

    // cierra objetos Statement y Connection
    public void desconectarDeBaseDatos() {
          if (conectadoABaseDatos) {
                // cierra objetos Statement y Connection
                try {
                      conjuntoResultados.close();
                      instruccion.close();
                      conexion.close();
                } // fin de try
                catch (SQLException excepcionSql) {
                      excepcionSql.printStackTrace();
                } // fin de catch
                finally // actualiza el estado de la conexión a la Base de Datos
                {
                      conectadoABaseDatos = false;
                } // fin de finally
          } // fin de if
    } // fin del método desconectarDeBaseDatos
} // fin de la clase ResultSetTableModel
```

Clase principal Main

```
public class EjecutarPruebaSwing {
    public static void main(String[] args) {
          new VentanaResultadosConsulta();
    }
}
```

## 6.5 Aplicación de altas, bajas, modificaciones y busqueda en una BD

Para que funcione el programa se debe tener la Base de Datos *Empleados* con su tabla *DatosContacto* creada con el administrador de Base de Datos MySQL o MariaDB, con algunos datos cargados y el servidor corriendo (`started`).

La aplicación muestra la siguiente ventana:


A continuación se muestra el código de la clase que maneja los datos de la BD donde se definen los métodos que realizan altas, bajas, modificaciones y búsqueda de datos.

La clase **`ConsultasPersona`** representa el Controlador dentro del MVC.

La interfaz **`PreparedStatement`** nos permite crear instrucciones SQL compiladas, que se ejecutan con más eficiencia que los objetos `Statement`. Las instrucciones PreparedStatement también pueden especificar parámetros, lo cual las hace más flexibles que las instrucciones Statement. Los programas pueden ejecutar la misma consulta varias veces, con distintos valores para los parámetros. Los signos de interrogación (?) en la instrucción SQL son receptáculos para valores que se pasarán como parte de la consulta en la base de datos. Antes de ejecutar una instrucción PreparedStatement, el programa debe especificar los valores de los parámetros mediante el uso de los métodos *`set`* de la interfaz `PreparedStatement`.

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.List;
import java.util.ArrayList;
import javax.swing.JOptionPane;

public class ConsultasPersona {
    static final String CONTROLADOR = "com.mysql.cj.jdbc.Driver";
    static final String URL_BASEDATOS = "jdbc:mysql://localhost/Empleados";
```

```
    private Connection conexion = null; // maneja la conexión
    private PreparedStatement seleccionarTodasLasPersonas = null;
    private PreparedStatement seleccionarPersonasPoridPersona = null;
    private PreparedStatement insertarNuevaPersona = null;
    private PreparedStatement borraridPersona = null;
    private PreparedStatement modificarDatosPersona = null;

    // constructor
    public ConsultasPersona() throws SQLException, ClassNotFoundException {
          try {
                Class.forName(CONTROLADOR);
                conexion = DriverManager.getConnection(URL_BASEDATOS, "root", "");

                // crea una consulta que selecciona todos los registros en la
tabla
                // DatosContacto
                seleccionarTodasLasPersonas = conexion.prepareStatement("SELECT *
FROM DatosContacto");

                // crea una consulta que selecciona las entradas con un id
                // específico
                seleccionarPersonasPoridPersona = conexion
                            .prepareStatement("SELECT * FROM DatosContacto WHERE
idPersona = ?");

                // crea instrucción insert para agregar una nueva entrada en la
Base
                // de Datos
                String insertar = "INSERT INTO `DatosContacto` (`idPersona`,
`Nombre`, `Apellido`, `Email`, `Telefono`) VALUES (NULL, ?, ?, ?, ?)";
                insertarNuevaPersona = conexion.prepareStatement(insertar);

                // crea instrucción delete para eliminar un registro en la Base de
                // Datos a partir del id
                String borrar = "DELETE FROM `DatosContacto` WHERE `idPersona`
= ?";
                this.borraridPersona = conexion.prepareStatement(borrar);

                // crea instrucción update para modificar un registro en la Base
de
                // Datos a partir del id
                String modificar = "UPDATE `DatosContacto` SET `Nombre`
= ? ,`Apellido` = ? ,`Email` = ?,`Telefono` = ? WHERE `idPersona` = ?";
                this.modificarDatosPersona = conexion.prepareStatement(modificar);
          } catch (SQLException excepcionSql) {
                // excepcionSql.printStackTrace();
                System.exit(1);
          }
    } // fin del constructor

    // Metodo que selecciona todos los registros de la tabla DatosContacto
    public List<Persona> obtenerTodasLasPersonas() {
```

```
		List<Persona> resultados = null;
		ResultSet conjuntoResultados = null;
		try {
			// executeQuery devuelve ResultSet que contiene las entradas que
			// coinciden
			conjuntoResultados = seleccionarTodasLasPersonas.executeQuery();

			resultados = new ArrayList<Persona>();

			while (conjuntoResultados.next()) {
				resultados.add(new Persona(conjuntoResultados.getInt(1),
conjuntoResultados.getString(2),
					conjuntoResultados.getString(3),
conjuntoResultados.getString(4),
					conjuntoResultados.getString(5)));

			}
			if (!conjuntoResultados.first())
				resultados = null;
		} catch (SQLException excepcionSql) {
			JOptionPane.showMessageDialog(null, "ERROR al intentar seleccionar
todas las personas");
			// excepcionSql.printStackTrace();
		} finally {
			try {
				conjuntoResultados.close();
			} catch (SQLException excepcionSql) {
				excepcionSql.printStackTrace();
				close();
			}
		}

		return resultados;
	} // fin del método obtenerTodasLasPersonas

	// Metodo que selecciona una persona por su Id
	public List<Persona> obtenerPersonasPoridPersona(int id) {
		List<Persona> resultados = null;
		ResultSet conjuntoResultados = null;

		try {
			seleccionarPersonasPoridPersona.setInt(1, id);

			// executeQuery devuelve ResultSet que contiene las entradas que
			// coinciden
			conjuntoResultados =
seleccionarPersonasPoridPersona.executeQuery();

			resultados = new ArrayList<Persona>();

			while (conjuntoResultados.next()) {
				resultados.add(new Persona(conjuntoResultados.getInt(1),
conjuntoResultados.getString(2),
```

```
                                    conjuntoResultados.getString(3),
conjuntoResultados.getString(4),
                                    conjuntoResultados.getString(5)));
                }
            } catch (SQLException excepcionSql) {
                // excepcionSql.printStackTrace();
            } finally {
                try {
                    conjuntoResultados.close();
                } catch (SQLException excepcionSql) {
                    // excepcionSql.printStackTrace();
                    close();
                }
            }

            return resultados;
    } // fin del método obtenerPersonasPoridPersona

    // Metodo para agregar un registro
    public int agregarPersona(String pnombre, String papellido, String pemail,
String ptelefono) {
            int resultado = 0;

            // establece los parámetros, después ejecuta insertarNuevaPersona
            try {
                insertarNuevaPersona.setString(1, pnombre);
                insertarNuevaPersona.setString(2, papellido);
                insertarNuevaPersona.setString(3, pemail);
                insertarNuevaPersona.setString(4, ptelefono);

                // inserta el nuevo registro; devuelve cant de filas actualizadas
                resultado = insertarNuevaPersona.executeUpdate();
            } catch (SQLException excepcionSql) {
                JOptionPane.showMessageDialog(null, "ERROR No pudo insertar nuevo
registro");
                excepcionSql.printStackTrace();
                // close();
            }

            return resultado;
    } // fin del método agregarPersona

    // Metodo que cierra la conexión a la Base de Datos
    public void close() {
            try {
                conexion.close();
            } catch (SQLException excepcionSql) {
                excepcionSql.printStackTrace();
            }
    } // fin del método close

    // Metodo para eliminar un registro
    public int borrarPersona(int id) {
```

```
            int resultado = 0;

            // establece los parámetros, después ejecuta insertarNuevaPersona
            try {
                  borraridPersona.setInt(1, id);

                  // elimina el registro; devuelve cant de filas actualizadas
                  resultado = borraridPersona.executeUpdate();
            } catch (SQLException excepcionSql) {
                  JOptionPane.showMessageDialog(null, "ERROR No pudo borrar
registro");
                  excepcionSql.printStackTrace();
                  // close();
            }

            return resultado;
      } // fin del método borrarPersona

      // Metodo para modificar un registro
      public int modificarPersona(int pid, String pnombre, String papellido, String
pemail, String ptelefono) {
            int resultado = 0;

            // establece los parámetros, después ejecuta insertarNuevaPersona
            try {
                  modificarDatosPersona.setInt(5, pid);
                  modificarDatosPersona.setString(1, pnombre);
                  modificarDatosPersona.setString(2, papellido);
                  modificarDatosPersona.setString(3, pemail);
                  modificarDatosPersona.setString(4, ptelefono);

                  // ejecuta el update de modificasion
                  resultado = modificarDatosPersona.executeUpdate();
            } catch (SQLException excepcionSql) {
                  JOptionPane.showMessageDialog(null, "ERROR No se pudo modificar el
registro");
                  excepcionSql.printStackTrace();
                  // close();
            }

            return resultado;
      } // fin del método modificarPersona
} // fin de la interfaz ConsultasPersona
```

El código que sigue corresponde a la clase ventana con los **`JButton`** para realizar las altas, bajas, modificaciones y consultas mostrando la información en los **`JTextField`**.

La clase **`VentanaEmpleados`** representa la vista dentro del MVC.

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.FlowLayout;
```

```
import java.awt.GridLayout;
import java.sql.SQLException;
import java.util.List;
import javax.swing.JButton;
import javax.swing.Box;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.BoxLayout;
import javax.swing.BorderFactory;
import javax.swing.JOptionPane;

public class VentanaEmpleados extends JFrame {
    private Persona entradaActual;
    private ConsultasPersona consultasPersona;
    private List<Persona> resultados;
    private int numeroDeEntradas = 0;
    private int indiceEntradaActual;

    private JButton botonExplorar;
    private JLabel etiquetaEmail;
    private JTextField campoTextoEmail;
    private JLabel etiquetaNombre;
    private JTextField campoTextoNombre;
    private JLabel etiquetaID;
    private JTextField campoTextoID;
    private JTextField campoTextoIndice;
    private JLabel etiquetaApellido;
    private JTextField campoTextoApellido;
    private JTextField campoTextoMax;
    private JButton botonSiguiente;
    private JLabel etiquetaDe;
    private JLabel etiquetaTelefono;
    private JTextField campoTextoTelefono;
    private JButton botonAnterior;
    private JButton botonConsulta;
    private JLabel etiquetaConsulta;
    private JPanel panelConsulta;
    private JPanel panelNavegar;
    private JPanel panelMostrar;
    private JTextField campoTextoConsulta;
    private JButton botonInsertar;
    private JButton botonGrabar;
    private JButton botonBorrar;
    private JButton botonModificar;
    private JButton botonGrabarModif;

    // constructor sin argumentos
    public VentanaEmpleados() throws SQLException, ClassNotFoundException {
          super("Datos de contacto de Empleados");

          // establece la conexión a la Base de Datos y las instrucciones
```

```
// PreparedStatement
consultasPersona = new ConsultasPersona();

// crea la GUI
panelNavegar = new JPanel();
botonAnterior = new JButton();
campoTextoIndice = new JTextField(2);
etiquetaDe = new JLabel();
campoTextoMax = new JTextField(2);
botonSiguiente = new JButton();
panelMostrar = new JPanel();
etiquetaID = new JLabel();
campoTextoID = new JTextField(10);
etiquetaNombre = new JLabel();
campoTextoNombre = new JTextField(10);
etiquetaApellido = new JLabel();
campoTextoApellido = new JTextField(10);
etiquetaEmail = new JLabel();
campoTextoEmail = new JTextField(10);
etiquetaTelefono = new JLabel();
campoTextoTelefono = new JTextField(10);
panelConsulta = new JPanel();
etiquetaConsulta = new JLabel();
campoTextoConsulta = new JTextField(10);
botonConsulta = new JButton();
botonExplorar = new JButton();
botonInsertar = new JButton();
botonGrabar = new JButton();
botonBorrar = new JButton();
botonModificar = new JButton();
botonGrabarModif = new JButton();

setLayout(new FlowLayout(FlowLayout.CENTER, 10, 10));
setSize(400, 300);
setResizable(true);

panelNavegar.setLayout(new BoxLayout(panelNavegar, BoxLayout.X_AXIS));

botonAnterior.setText("Anterior");
botonAnterior.setEnabled(false);
botonAnterior.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent evt) {
            botonAnteriorActionPerformed(evt);
      } // fin del método actionPerformed
} // fin de la clase interna anónima
); // fin de la llamada a addActionListener

panelNavegar.add(botonAnterior);
panelNavegar.add(Box.createHorizontalStrut(10));

campoTextoIndice.setHorizontalAlignment(JTextField.CENTER);
campoTextoIndice.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent evt) {
```

```
                campoTextoIndiceActionPerformed(evt);
            } // fin del método actionPerformed
        } // fin de la clase interna anónima
        ); // fin de la llamada a addActionListener

        panelNavegar.add(campoTextoIndice);
        panelNavegar.add(Box.createHorizontalStrut(10));

        etiquetaDe.setText("de");
        panelNavegar.add(etiquetaDe);
        panelNavegar.add(Box.createHorizontalStrut(10));

        campoTextoMax.setHorizontalAlignment(JTextField.CENTER);
        campoTextoMax.setEditable(false);
        panelNavegar.add(campoTextoMax);
        panelNavegar.add(Box.createHorizontalStrut(10));

        botonSiguiente.setText("Siguiente");
        botonSiguiente.setEnabled(false);
        botonSiguiente.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent evt) {
                botonSiguienteActionPerformed(evt);
            } // fin del método actionPerformed
        } // fin de la clase interna anónima
        ); // fin de la llamada a addActionListener

        panelNavegar.add(botonSiguiente);
        add(panelNavegar);

        panelMostrar.setLayout(new GridLayout(5, 2, 4, 4));

        etiquetaID.setText("ID Empleado:");
        panelMostrar.add(etiquetaID);

        campoTextoID.setEditable(false);
        panelMostrar.add(campoTextoID);

        etiquetaNombre.setText("Nombre:");
        panelMostrar.add(etiquetaNombre);
        panelMostrar.add(campoTextoNombre);

        etiquetaApellido.setText("Apellido:");
        panelMostrar.add(etiquetaApellido);
        panelMostrar.add(campoTextoApellido);

        etiquetaEmail.setText("Email:");
        panelMostrar.add(etiquetaEmail);
        panelMostrar.add(campoTextoEmail);

        etiquetaTelefono.setText("Telefono:");
        panelMostrar.add(etiquetaTelefono);
        panelMostrar.add(campoTextoTelefono);
        add(panelMostrar);
```

```
        panelConsulta.setLayout(new BoxLayout(panelConsulta, BoxLayout.X_AXIS));

        panelConsulta.setBorder(BorderFactory.createTitledBorder("Buscar un
registro por ID"));
        etiquetaConsulta.setText("idEmpleado:");
        panelConsulta.add(Box.createHorizontalStrut(5));
        panelConsulta.add(etiquetaConsulta);
        panelConsulta.add(Box.createHorizontalStrut(10));
        panelConsulta.add(campoTextoConsulta);
        panelConsulta.add(Box.createHorizontalStrut(10));

        botonConsulta.setText("Buscar");
        botonConsulta.addActionListener(new ActionListener() {
              public void actionPerformed(ActionEvent evt) {
                    botonConsultaActionPerformed(evt);
              } // fin del método actionPerformed
        } // fin de la clase interna anónima
        ); // fin de la llamada a addActionListener

        panelConsulta.add(botonConsulta);
        panelConsulta.add(Box.createHorizontalStrut(5));
        add(panelConsulta);

        botonExplorar.setText("Ver todos");
        botonExplorar.addActionListener(new ActionListener() {
              public void actionPerformed(ActionEvent evt) {
                    botonExplorarActionPerformed(evt);
              } // fin del método actionPerformed
        } // fin de la clase interna anónima
        ); // fin de la llamada a addActionListener

        add(botonExplorar);

        botonInsertar.setText("Insertar");
        botonInsertar.addActionListener(new ActionListener() {
              public void actionPerformed(ActionEvent evt) {
                    botonInsertarActionPerformed(evt);
              } // fin del método actionPerformed
        } // fin de la clase interna anónima
        ); // fin de la llamada a addActionListener

        add(botonInsertar);
        ////////////////////////
        botonGrabar.setText("Grabar");
        botonGrabar.addActionListener(new ActionListener() {
              public void actionPerformed(ActionEvent evt) {
                    botonGrabarActionPerformed(evt);
              } // fin del método actionPerformed
        } // fin de la clase interna anónima
        ); // fin de la llamada a addActionListener

        add(botonGrabar);
```

```
        botonGrabar.setVisible(false);
        //////////////
        botonBorrar.setText("Eliminar");
        botonBorrar.addActionListener(new ActionListener() {
              public void actionPerformed(ActionEvent evt) {
                    botonBorrarActionPerformed(evt);
              } // fin del método actionPerformed
        } // fin de la clase interna anónima
        ); // fin de la llamada a addActionListener
        add(botonBorrar);
        ///////////////
        botonModificar.setText("Modificar");
        botonModificar.addActionListener(new ActionListener() {
              public void actionPerformed(ActionEvent evt) {
                    botonModificarActionPerformed(evt);
              } // fin del método actionPerformed
        } // fin de la clase interna anónima
        ); // fin de la llamada a addActionListener
        add(botonModificar);
        //////////
        botonGrabarModif.setText("Grabar modif");
        botonGrabarModif.addActionListener(new ActionListener() {
              public void actionPerformed(ActionEvent evt) {
                    botonGrabarModifActionPerformed(evt);
              } // fin del método actionPerformed
        } // fin de la clase interna anónima
        ); // fin de la llamada a addActionListener
        add(botonGrabarModif);
        botonGrabarModif.setVisible(false);

        ///////////////
        addWindowListener(new WindowAdapter() {
              public void windowClosing(WindowEvent evt) {
                    consultasPersona.close(); // cierra la conexión a la Base de
                                                    // Datos
                    System.exit(0);
              } // fin del método windowClosing
        } // fin de la clase interna anónima
        ); // fin de la llamada a addWindowListener

        botonExplorarActionPerformed(null);
        setVisible(true);
  } // fin del constructor sin argumentos

  // maneja la llamada cuando se hace click en botonAnterior
  private void botonAnteriorActionPerformed(ActionEvent evt) {
        indiceEntradaActual--;

        if (indiceEntradaActual < 0)
              indiceEntradaActual = numeroDeEntradas - 1;

        campoTextoIndice.setText("" + (indiceEntradaActual + 1));
        campoTextoIndiceActionPerformed(evt);
```

```
    } // fin del método botonAnteriorActionPerformed

    // maneja la llamada cuando se hace click en botonSiguiente
    private void botonSiguienteActionPerformed(ActionEvent evt) {
          indiceEntradaActual++;

          if (indiceEntradaActual >= numeroDeEntradas)
                indiceEntradaActual = 0;

          campoTextoIndice.setText("" + (indiceEntradaActual + 1));
          campoTextoIndiceActionPerformed(evt);
    } // fin del método botonSiguienteActionPerformed

    // maneja la llamada cuando se hace click en botonConsulta
    private void botonConsultaActionPerformed(ActionEvent evt) {
          String idPersona = campoTextoConsulta.getText();
          if (idPersona.equals("")) {
                JOptionPane.showMessageDialog(this, "Debe ingresar un nro de ID");
                campoTextoConsulta.requestFocusInWindow();
          } else {
                resultados =
consultasPersona.obtenerPersonasPoridPersona(Integer.parseInt(campoTextoConsulta.
getText()));
                numeroDeEntradas = resultados.size();

                if (numeroDeEntradas != 0) {
                      indiceEntradaActual = 0;
                      entradaActual = resultados.get(indiceEntradaActual);
                      campoTextoID.setText("" + entradaActual.getidPersona());
                      campoTextoNombre.setText(entradaActual.getNombre());
                      campoTextoApellido.setText(entradaActual.getApellido());
                      campoTextoEmail.setText(entradaActual.getEmail());
                      campoTextoTelefono.setText(entradaActual.getTelefono());
                      campoTextoMax.setText("" + numeroDeEntradas);
                      campoTextoIndice.setText("" + (indiceEntradaActual + 1));
                      botonSiguiente.setEnabled(true);
                      botonAnterior.setEnabled(true);
                } // end if
                else {
                      campoTextoID.setText("");
                      campoTextoNombre.setText("");
                      campoTextoApellido.setText("");
                      campoTextoEmail.setText("");
                      campoTextoTelefono.setText("");
                      campoTextoMax.setText("0");
                      campoTextoIndice.setText("0");
                      botonSiguiente.setEnabled(true);
                      botonAnterior.setEnabled(true);
                      // botonExplorarActionPerformed( evt );
                }
                campoTextoConsulta.requestFocusInWindow();
          }
    } // fin del método botonConsultaActionPerformed
```

```
    // maneja la llamada cuando se introduce un nuevo valor en campoTextoIndice
    private void campoTextoIndiceActionPerformed(ActionEvent evt) {
          indiceEntradaActual = (Integer.parseInt(campoTextoIndice.getText()) -
1);

          if (numeroDeEntradas != 0 && indiceEntradaActual < numeroDeEntradas) {
                entradaActual = resultados.get(indiceEntradaActual);
                campoTextoID.setText("" + entradaActual.getidPersona());
                campoTextoNombre.setText(entradaActual.getNombre());
                campoTextoApellido.setText(entradaActual.getApellido());
                campoTextoEmail.setText(entradaActual.getEmail());
                campoTextoTelefono.setText(entradaActual.getTelefono());
                campoTextoMax.setText("" + numeroDeEntradas);
                campoTextoIndice.setText("" + (indiceEntradaActual + 1));
          } // fin de if
    } // fin del método campoTextoIndiceActionPerformed

    // maneja la llamada cuando se hace clic en botonExplorar
    private void botonExplorarActionPerformed(ActionEvent evt) {
          try {
                resultados = consultasPersona.obtenerTodasLasPersonas();
                if (resultados == null)
                      numeroDeEntradas = 0;
                else
                      numeroDeEntradas = resultados.size();

                if (numeroDeEntradas != 0) {
                      indiceEntradaActual = 0;
                      entradaActual = resultados.get(indiceEntradaActual);
                      campoTextoID.setText("" + entradaActual.getidPersona());
                      campoTextoNombre.setText(entradaActual.getNombre());
                      campoTextoApellido.setText(entradaActual.getApellido());
                      campoTextoEmail.setText(entradaActual.getEmail());
                      campoTextoTelefono.setText(entradaActual.getTelefono());
                      campoTextoMax.setText("" + numeroDeEntradas);
                      campoTextoIndice.setText("" + (indiceEntradaActual + 1));
                      botonSiguiente.setEnabled(true);
                      botonAnterior.setEnabled(true);
                }
          } catch (Exception e) {
                e.printStackTrace();
          }
    } // fin del método botonExplorarActionPerformed

    // maneja la llamada cuando se hace click en botonInsertar
    private void botonInsertarActionPerformed(ActionEvent evt) {
          campoTextoID.setEditable(false);
          campoTextoID.setText("");
          campoTextoNombre.setText("");
          campoTextoApellido.setText("");
          campoTextoEmail.setText("");
          campoTextoTelefono.setText("");
```

```
            JOptionPane.showMessageDialog(null, "Para agregar debe llenar al menos
el campo Apellido");
            this.campoTextoApellido.requestFocusInWindow();
            this.botonInsertar.setVisible(false);
            this.botonGrabar.setText("Grabar datos del nuevo registro");
            this.panelConsulta.setVisible(false);
            this.botonGrabar.setVisible(true);
            this.botonBorrar.setVisible(false);
            this.botonModificar.setVisible(false);
            this.botonExplorar.setVisible(false);

            // botonExplorarActionPerformed( evt );
      } // fin del método botonInsertarActionPerformed

      private void botonGrabarActionPerformed(ActionEvent evt) {
            if (campoTextoApellido.getText().equals(""))
                  JOptionPane.showMessageDialog(null, "No se agrego ningun registro,
el campo Apellido no fue completado");
            else {
                  int resultado =
consultasPersona.agregarPersona(campoTextoNombre.getText().toString(),
                              campoTextoApellido.getText().toString(),
campoTextoEmail.getText().toString(),
                              campoTextoTelefono.getText().toString());
                  String persona = campoTextoNombre.getText().toString() + " " +
campoTextoApellido.getText().toString();

                  if (resultado == 1)
                        JOptionPane.showMessageDialog(this, "Se agrego al Empleado"
+ persona, "Se agrego Empleado",
                                    JOptionPane.PLAIN_MESSAGE);
                  else
                        JOptionPane.showMessageDialog(this, "No se agrego al
Empleado!", "ERROR", JOptionPane.PLAIN_MESSAGE);
            }
            this.panelConsulta.setVisible(true);
            this.botonGrabar.setText("Grabar");
            this.botonGrabar.setVisible(false);
            this.botonExplorar.setVisible(true);
            this.botonInsertar.setVisible(true);
            this.botonBorrar.setVisible(true);
            this.botonModificar.setVisible(true);

            botonExplorarActionPerformed(evt);
      } // fin botonGrabarActionPerformed

      private void botonBorrarActionPerformed(ActionEvent evt) {
            if (this.campoTextoID.getText().equals(""))
                  JOptionPane.showMessageDialog(null, "No se elimino ningun
registro, el campo ID esta vacio");
            else {
                  String persona = this.campoTextoNombre.getText() + " " +
this.campoTextoApellido.getText();
```

```
                int resp = JOptionPane.showConfirmDialog(this, "Esta seguro de que
quiere eliminar a " + persona);
                if (resp == JOptionPane.YES_OPTION) {
                      int resultado =
consultasPersona.borrarPersona(Integer.parseInt(campoTextoID.getText().toString()
));
                      if (resultado == 1)
                            JOptionPane.showMessageDialog(this, "Se elimino a " +
persona, "Se elimino Empleado",
                                        JOptionPane.PLAIN_MESSAGE);
                      else
                            JOptionPane.showMessageDialog(this, "No se elimino el
Empleado!", "ERROR",
                                        JOptionPane.PLAIN_MESSAGE);
                } else
                      JOptionPane.showMessageDialog(this, "No se elimino a " +
persona);
          }
          botonExplorarActionPerformed(evt);
    }

    private void botonModificarActionPerformed(ActionEvent evt) {
          this.botonModificar.setVisible(false);
          this.botonGrabarModif.setVisible(true);
          this.botonInsertar.setVisible(false);
          this.botonBorrar.setVisible(false);
          this.botonModificar.setVisible(false);
          this.botonExplorar.setVisible(false);
          this.campoTextoApellido.requestFocusInWindow();
    }

    private void botonGrabarModifActionPerformed(ActionEvent evt) {
          if (campoTextoApellido.getText().equals(""))
                JOptionPane.showMessageDialog(null, "No se modifico ningun
registro, el campo ID no fue completado");
          else {
                int resultado =
consultasPersona.modificarPersona(Integer.parseInt(campoTextoID.getText().toStrin
g()),
                            campoTextoNombre.getText().toString(),
campoTextoApellido.getText().toString(),
                            campoTextoEmail.getText().toString(),
campoTextoTelefono.getText().toString());
                String persona = campoTextoNombre.getText().toString() + " " +
campoTextoApellido.getText().toString();

                if (resultado == 1)
                      JOptionPane.showMessageDialog(this, "Se modificaron los
datos del Empleado " + persona,
                                  "Se agrego persona", JOptionPane.PLAIN_MESSAGE);
                else
                      JOptionPane.showMessageDialog(this, "No se modificaron los
datos del Empleado!", "ERROR",
```

```
                                    JOptionPane.PLAIN_MESSAGE);
            }
            this.botonGrabar.setVisible(false);
            this.botonGrabarModif.setVisible(false);
            this.botonExplorar.setVisible(true);
            this.botonModificar.setVisible(true);
            this.botonInsertar.setVisible(true);
            this.botonBorrar.setVisible(true);
            botonExplorarActionPerformed(evt);
    } // fin botonGrabarModifActionPerformed
} // fin de la clase VentanaEmpleados
```

Clase **Persona**

```
public class Persona {
    private int idPersona;
    private String nombre;
    private String apellido;
    private String email;
    private String telefono;

    public Persona(int idPersona, String Nombre, String Apellido, String Email,
String Telefono) {
        this.idPersona = idPersona;
        this.nombre = Nombre;
        this.apellido = Apellido;
        this.email = Email;
        this.telefono = Telefono;
    }

    public String getTelefono() {
        return telefono;
    }

    public void setTelefono(String Telefono) {
        this.telefono = Telefono;
    }

    public String getApellido() {
        return apellido;
    }

    public void setApellido(String Apellido) {
        this.apellido = Apellido;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String Email) {
        this.email = Email;
```

```
    }

    public int getidPersona() {
        return idPersona;
    }

    public void setidPersona(int idPersona) {
        this.idPersona = idPersona;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String Nombre) {
        this.nombre = Nombre;
    }
}
```

Clase principal **TestABMC**

```
import java.sql.SQLException;

public class TestABMC {

    public static void main(String[] args) throws ClassNotFoundException,
SQLException {
          new VentanaEmpleados();
    }
}
```

## 6.6 Metadatos de una Base de Datos

Hay casos en los que se requiere conocer la estructura de una base de datos (nombre y diseño de las tablas, tipos de los campos, etc.). Los datos que describen la estructura de las bases de datos es lo que se conoce como metadatos.

Los metadatos se obtienen utilizando el método *getMetaData* de la clase **Connection**, por lo que es el objeto de la conexión el que permite obtener estos metadatos. El resultado de este método es un objeto de clase **DatabaseMetaData**.

```
DatabaseMetaData metadatos = conexion.getMetaData();
```

Una vez obtenido el objeto se pueden utilizar estos métodos para obtener información sobre la base de datos:

| Método de DatabaseMetadata | Uso |
|---|---|
| `ResultSet getColumns(String catálogo, String plantillaEsquema, String` | Obtiene una tabla de resultados (ResultSet) en la que cada fila es un campo que cumple el nombre de campo indicado en la |

| | |
|---|---|
| `plantillaTabla, String plantillaCampo)` | plantilla de campo (que puede ser null). |
| `Connection getConnection()` | Devuelve el objeto Connection relacionado con este objeto |
| `String getDatabaseProductName()` | Devuelve el nombre comercial del sistema gestor de base de datos en uso |
| `String getDatabaseProductVersion()` | Devuelve la versión de producto del sistema de base de datos en uso |
| `String getDriverName()` | Devuelve el nombre del driver JDBC en uso |
| `ResultSet getIndexInfo(String catalog, String schema, String tabla, boolean única, boolean aproximación)` | Obtiene un conjunto de resultados que describe los índices de la tabla indicada. |
| `String getURL()` | Obtiene una cadena con el URL del sistema gestor de bases de datos. |
| `String getUserName()` | Devuelve el nombre de usuario actual del gestor de bases de datos. |

El método *getMetaData* de la clase **ResultSet** da como resultado un objeto de tipo ResultSetMetaData que devuelve información de control del conjunto de resultados.

Sus métodos más interesantes son:

| Método de ResultSetMetaData | Uso |
|---|---|
| `int getColumnCount()` | Obtiene el número de columnas del conjunto de resultados |
| `int getColumnDisplaySize(int númeroDeColumna)` | Indica la anchura que se destina en pantalla a mostrar el contenido de la columna indicada |
| `String getColumnName(int nColumna)` | Devuelve el nombre de la columna con el número indicado |
| `int getColumnType(int nColumns)` | Obtiene el tipo de datos SQL estándar de la columna indicada |
| `int getColumnTypeName(int nColumna)` | Obtiene el tipo de datos compatible con la base de datos en uso de la columna indicada |
| `int getPrecission(int nColumna)` | Devuelve la precisión de decimales de la columna dada |
| `int getScale(int nColumna)` | Obtiene el número de decimales que se muestran de la columna |
| `boolean isAutoincrement(int nColumna)` | Indica si el campo es autoincrementable. |
| `boolean isCaseSensitive(int nColumna)` | Indica si la columna distingue entre mayúsculas y minúsculas |
| `boolean isReadOnly(int nColumna)` | Indica si el campo es de sólo lectura. |
| `boolean isSearchable(int nColumna)` | indica si la columna puede figurar en el apartado WHERE de una consulta SELECT |
| `boolean isSigned(int nColumna)` | Indica si la columna posee números con signo |

## 6.7 PROCEDIMIENTOS ALMACENADOS

Muchos sistemas de administración de bases de datos pueden almacenar instrucciones de SQL individuales o conjuntos de instrucciones de SQL en una base de datos, para que los programas que tengan acceso a esa base de datos puedan invocar esas instrucciones. A dichas instrucciones de SQL se les conoce como procedimientos almacenados **(stored procedure)**. JDBC permite a los programas invocar procedimientos almacenados mediante el uso de objetos que implementen a la interfaz

**`CallableStatement`**. Los objetos `CallableStatement` pueden recibir argumentos especificados con los métodos heredados de la interfaz **`PreparedStatement`**. Además, los objetos `CallableStatement` pueden especificar parámetros de salida, en los cuales un procedimiento almacenado puede colocar valores de retorno. La interfaz **`CallableStatement`** incluye métodos para especificar cuáles parámetros en un procedimiento almacenado son parámetros de salida. La interfaz también incluye métodos para obtener los valores de los parámetros de salida devueltos de un procedimiento almacenado.

## 6.8 PROCESAMIENTO DE TRANSACCIONES

Muchas aplicaciones de bases de datos requieren garantías de que una serie de operaciones de inserción, actualización y eliminación se ejecuten de manera apropiada, antes de que la aplicación continúe procesando la siguiente operación en la base de datos. Por ejemplo, al transferir dinero por medios electrónicos entre dos cuentas de banco, varios factores determinan si la transacción fue exitosa. Empezamos por especificar la cuenta de origen y el monto que deseamos transferir de esa cuenta hacia una cuenta de destino. Después, especificamos la cuenta de destino. El banco comprueba la cuenta de origen para determinar si hay suficientes fondos en ella como para poder completar la transferencia. De ser así, el banco retira el monto especificado de la cuenta de origen y, si todo sale bien, deposita el dinero en la cuenta de destino para completar la transferencia. ¿Qué ocurre si la transferencia falla después de que el banco retira el dinero de la cuenta de origen? En un sistema bancario apropiado, el banco vuelve a depositar el dinero en la cuenta de origen. ¿Cómo se sentiría usted si el dinero se restara de su cuenta de origen y el banco *no* depositara el dinero en la cuenta de destino?

El **procesamiento de transacciones** permite a un programa que interactúa con una base de datos tratar una operación en la base de datos (o un conjunto de operaciones) como una sola operación. Dicha operación también se conoce como **operación atómica** o **transacción**. Al final de una transacción, se puede tomar una de dos decisiones: **confirmar (commit) la transacción** o **rechazar (roll back) la transacción**. Al confirmar la transacción se finaliza(n) la(s) operación(es) de la base de datos; todas las inserciones, actualizaciones y eliminaciones que se llevaron a cabo como parte de la transacción no pueden invertirse sin tener que realizar una nueva operación en la base de datos. Al rechazar la transacción, la base de datos queda en el estado anterior a la operación. Esto es útil cuando una parte de una transacción no se completa en forma apropiada. En nuestra discusión acerca de la transferencia entre cuentas bancarias, la transacción se rechazaría si el depósito no pudiera realizarse en la cuenta de destino.

JAVA ofrece el procesamiento de transacciones a través de varios métodos de la interfaz **`Connection`**. El método *`setAutoCommit`* especifica si cada instrucción SQL se confirma una vez completada (un argumento **`true`**), o si deben agruparse varias instrucciones SQL para formar una transacción (un argumento **`false`**). Si el argumento para *`setAutoCommit`* es **`false`**, el programa debe colocar después de la última instrucción SQL en la transacción una llamada al método *`commit`* de **`Connection`** (para confirmar los cambios en la base de datos) o al método *`rollback`* de **`Connection`** (para regresar la base

de datos al estado anterior a la transacción). La interfaz **`Connection`** también cuenta con el método *`getAutoCommit`* para determinar el estado de autoconfirmación para el objeto `Connection`.

## 6.9 Proceso por lotes

Una mejora importante de JDBC 2 es la facultad de procesar múltiples instrucciones SQL mediante lotes (*Batch*). Los procesos por lotes permiten recopilar y lanzar una serie larga de instrucciones.

Esto se realizar mediante los métodos *`addBatch`* y *`executeBatch`* de la clase **`Statement`**. El primero permite añadir nuevas instrucciones al proceso por lotes. El segundo lanza las instrucciones almacenadas. El resultado de *`executeBatch`* es un array de enteros donde cada elemento es el número de filas modificadas por la acción lanzada correspondiente. Es decir si el primer comando SQL del proceso modifica tres filas y el segundo seis, el resultado es un array de dos elementos donde el primero vale tres y el segundo seis. Sólo se permiten colocar instrucciones SQL de actualización (UPDATE, INSERT, CREATE TABLE, DELETE,..). El método *`clearBatch`* permite borrar el contenido del proceso por lotes (de hecho borra todas las consultas del *`Statement`*).

## BIBLIOGRAFÍA


- Ceballos, Fco. Javier, “JAVA 2 Curso de programación” 4ta Ed. (Ra-Ma 2010)
- Deitel, Paul y Deitel, Harvey, “JAVA Cómo programar” 9na Ed. (Pearson 2012)
- Kuhn, Mónica, “Apuntes de Programación II” INSPT/UTN (2014)
- Sánchez, Jorge, “JAVA 2” (2004)


## LICENCIA



Matías E. García

Prof. & Tec. en Informática Aplicada
www.profmatiasgarcia.com.ar
info@profmatiasgarcia.com.ar