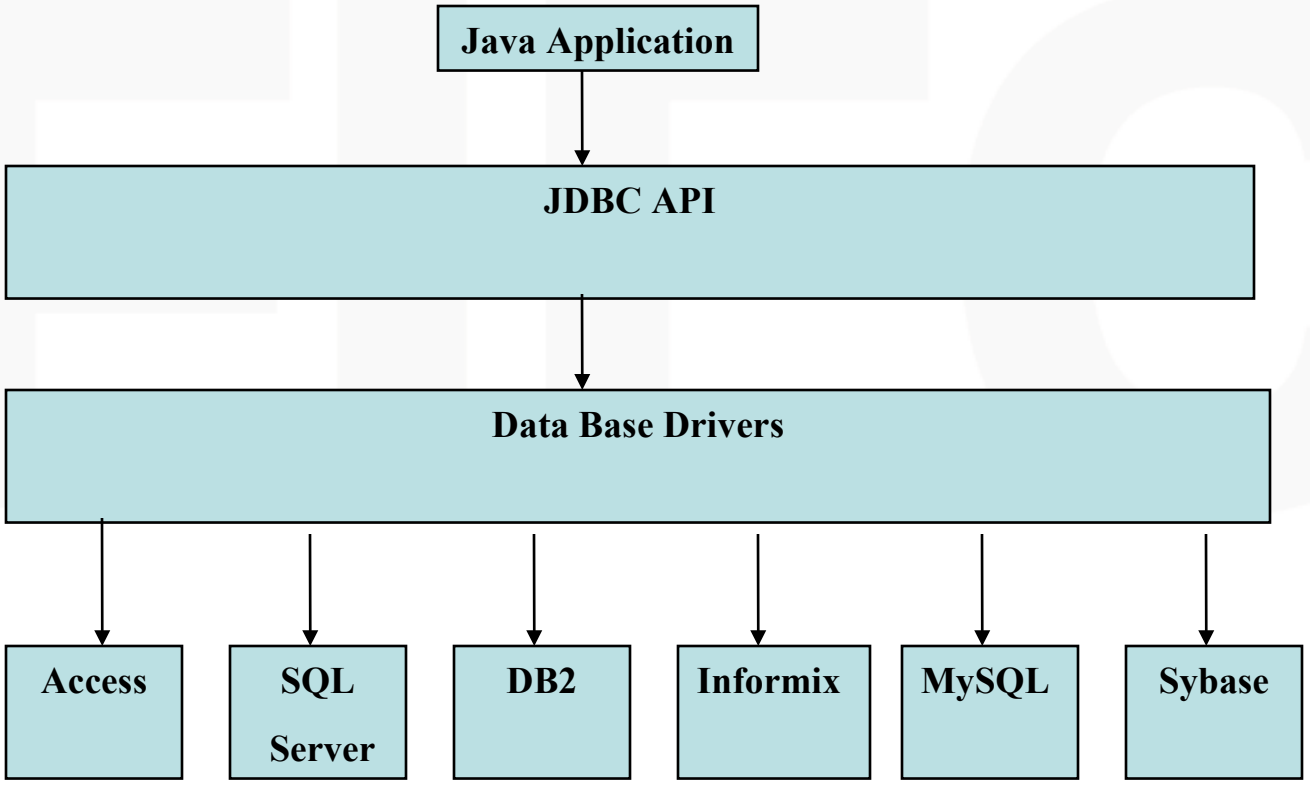
Java y Bases de Datos

JDBC

- JDBC (Java DataBase Connectivity) es la tecnología Java que permite a las aplicaciones interactuar directamente con motores de base de datos relacionales.
- La **API JDBC** es una parte integral de la plataforma Java, por lo tanto no es necesario descargar ningún paquete adicional para usarla.
- JDBC provee una interfase única, que independiza a las aplicaciones del motor de la base de datos.
- Un **driver JDBC** es usado por la JVM para traducir las invocaciones JDBC en invocaciones que la base de datos entiende.

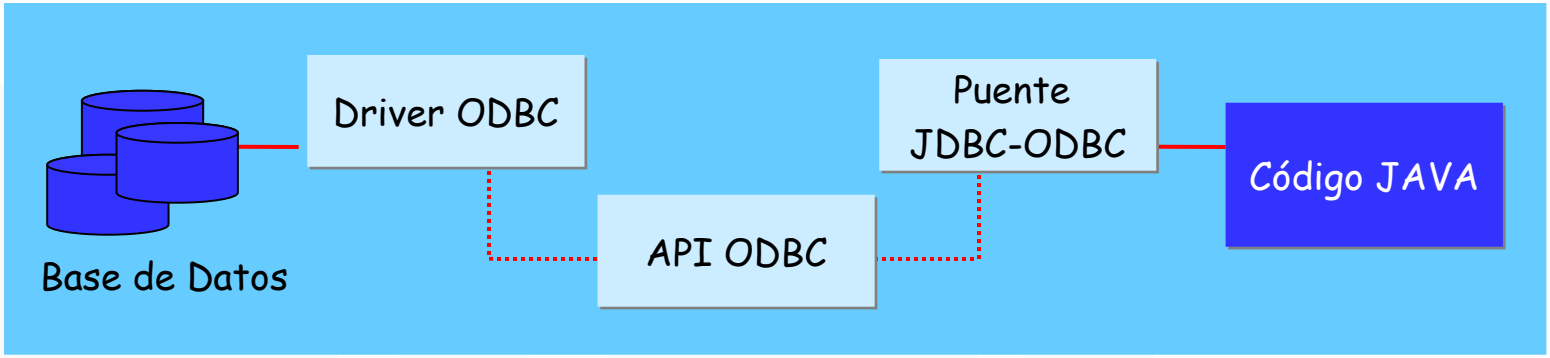
Arquitectura JDBC



Drivers JDBC

- Existen drivers JDBC para la mayoría de los motores de base de datos. Típicamente, los fabricantes de bases de datos proveen el driver JDBC para su motor.
- Es posible encontrarlos en Internet:
  - http://industry.java.sun.com/products/jdbc/drivers
- Hay cuatro tipos de JDBC drivers:
  - Tipo 1: JDBC-ODBC Bridge
  - Tipo 2: Native API, partially java
  - Tipo 3: JDBC Network Driver
  - Tipo 4: 100% Java

Driver Tipo 1 (JDBC-ODBC Bridge)

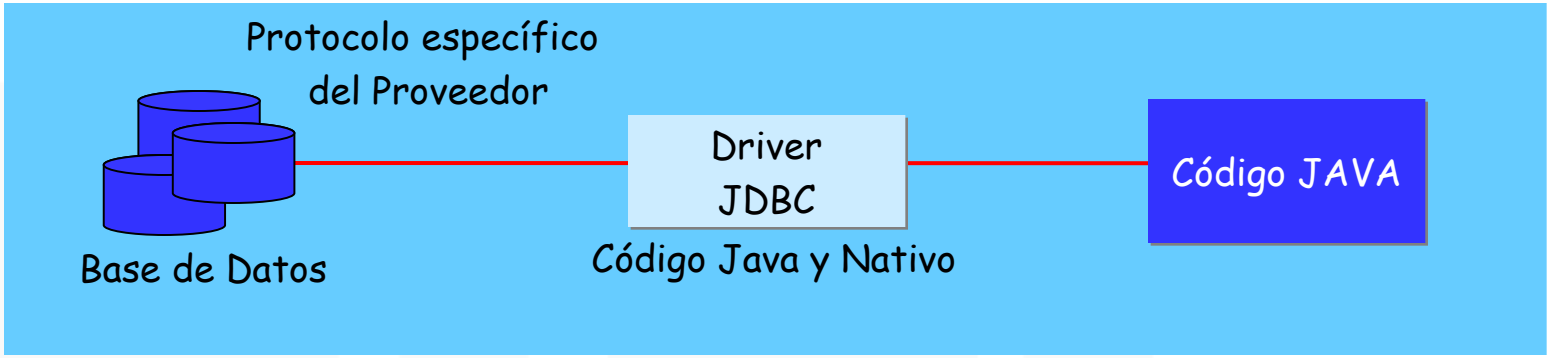


- El driver JDBC-ODBC es parte de la plataforma Java. No es un driver 100 % Java.
- Traduce invocaciones JDBC a invocaciones ODBC a través de librerías ODBC del sistema operativo.
- No es una solución buena, pero en algunas situaciones es la única, tal es el caso de Microsoft Access.

Desventajas

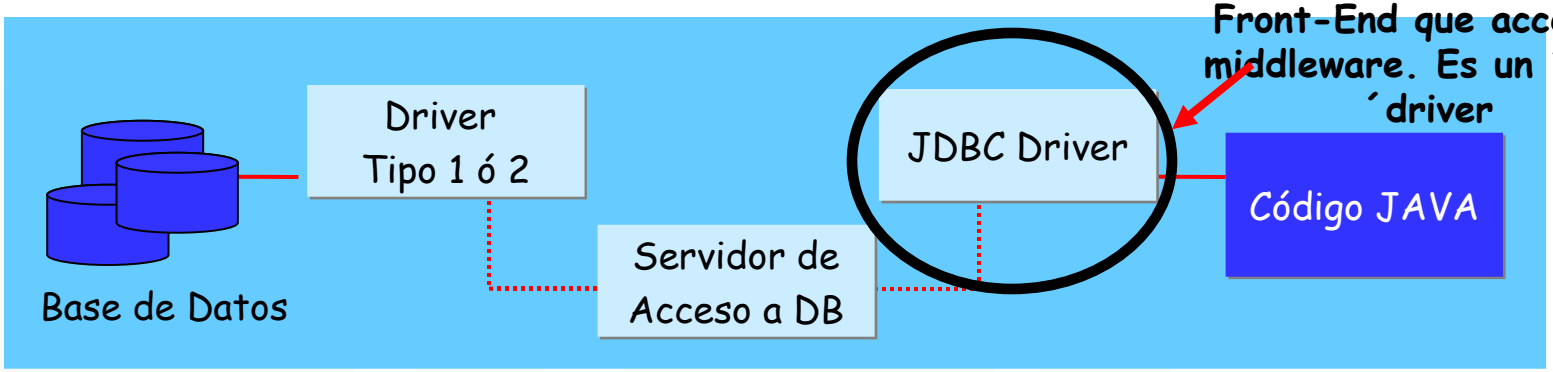
- Se requieren múltiples capas de software para hacer las llamadas a la Base de Datos.
- Se requiere la instalación de software adicional (configuración ODBC).

Driver Tipo 2 (Native API, partially java)



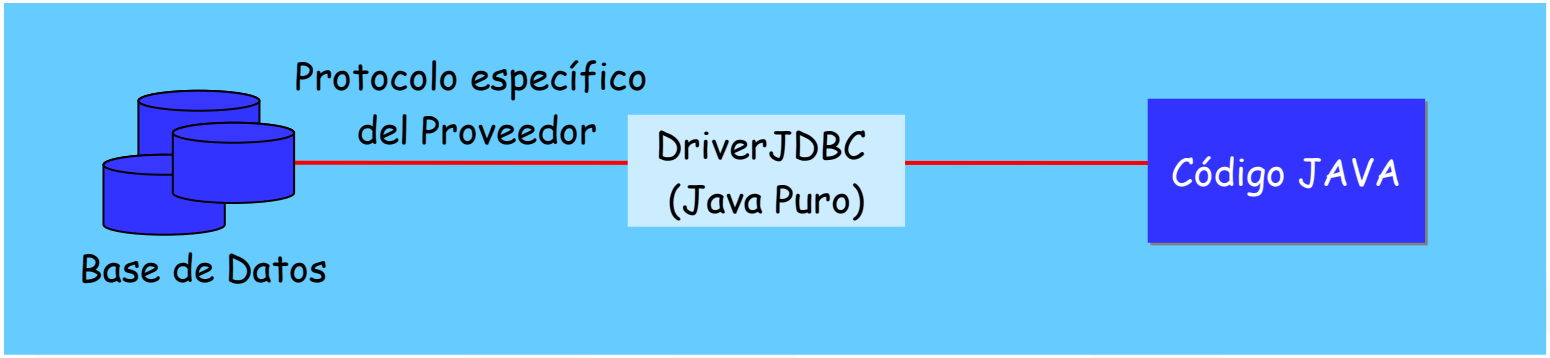
- Conceptualmente es similar al driver de tipo 1, excepto que se usa una capa menos (no está la capa de traducción ODBC). No es un driver 100 % Java.
- Cuando se realiza una invocación a la base de datos, a través de JBDC, el driver traduce el requerimiento en algo que la API del fabricante de la base de datos entiende.
- La base de datos, procesa el requerimiento y devuelve el resultado a través de la API que lo reenvía al driver. El driver formatea el resultado al estándar JDBC y lo devuelve al programa.

Driver Tipo 3 (JDBC Network Driver)



- Actúa como un front-end para acceso a servidores de base de datos.
- El programa envía una invocación JDBC a través del ´proxy´ driver, quien lo envía a la capa intermedia o *middleware*, sin traducción.
- El *middleware* completa el requerimiento usando otro driver JDBC.
- El *middleware* habla con la base de datos, a través de un driver Tipo 1 o 2.
- Requiere de la instalación de un middleware.

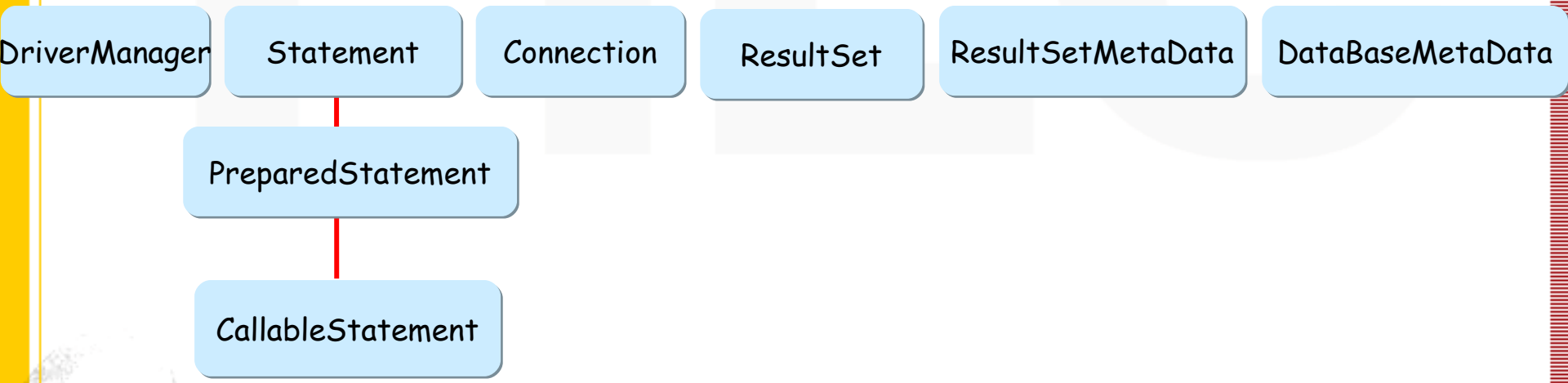
Driver Tipo 4



- Es un driver **Java Puro,** que habla directamente con la base de datos.
- Es el método más eficiente de acceso a bases de datos.
- No requiere de ninguna librería adicional ni de la instalación de un *middleware*, con lo cual es de *deployment* más simple.
- La mayoría de los fabricantes, proveen drivers JDBC de tipo 4 para sus bases de datos.

La API JDBC

- Las clases e interfases de la API JDBC están en el paquete **java.sql**.
- En estas se encuentran definidos métodos que permiten: conectarse y recuperar información de la BD.



Conexión a la Base de Datos

- La conexión se establece a través del driver, que se carga en ejecución mediante el método:
  - **Class.forName(String nombredelDriver)**
- Una vez cargado el driver, la conexión a la BD se realiza invocando a alguno de los siguientes métodos de la clase **java.sql.DriverManager** :
  - public static Connection getConnection(String url)
  - public static Connection getConnection(String url, java.util.Properties info)
  - public static Connection getConnection(String url, String usr, String pwd)

Ejemplo de una conexión a una Base de Datos

Esta conexión será usada para realizar todas las operaciones sobre la Base de Datos.

```
. . .
Connection miConexion;
try {
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    miConexion = DriverManager.getConnection("jdbc:odbc:empleadosDB");
      . . .
} catch (ClassNotFoundException e1) {
// captura el error: "no se encontró el driver"
} catch (SQLException e2) {
// captura el error: "no se pudo conectar a la BD"
}
```

Nombre del Driver

url de la BD

Origen de datos

Se deben manejar por lo menos dos excepciones. Una es para controlar si el Driver no es encontrado y la otra para verificar si se realizó la conexión.

Ejecución de Sentencias SQL

- Para realizar una consulta SQL a la BD se requiere de la creación de un objeto: **Statement**, **PreparedStatement** o **CallableStatement** usando uno de los métodos de **Connection**.
  - Statement createStatement()
  - Statement createStatement(int resultSetType, int resultSetConcurrency)
  - PreparedStatement preparedStatement(String sql)
  - PreparedStatement preparedStatement(String sql, int resultSetType, int resultSetConcurrency)
  - CallableStatement prepareCall(String sql)
  - CallableStatement prepareCall(String sql, int resultSetType, int resultSetConcurrency)

PreparedStatement

- Una **sentencia preparada** (PreparedStatement) es un tipo de sentencia que mejora la performance de las consultas.
- Una **sentencia preparada** se precompila antes de ser usada. La versión precompilada acepta diferente número de parámetros.
- A diferencia de las sentencias tradicionales cuando se crean requieren de la sentencia SQL como argumento del constructor. Esta sentencia es enviada al motor de BD para su compilación y cuando se ejecuta el motor de BD la corre sin previa compilación.
- Las sentencias preparadas manejan parámetros, con lo cual pueden ejecutarse muchas veces con distintos parámetros.
- **Ejemplo:**
  - PreparedStatement p_sent = miConexion.prepareStatement("SELECT * FROM Empleados WHERE edad = ?")

    p_sent.setInt(1, 55); //Se setea el parámetro

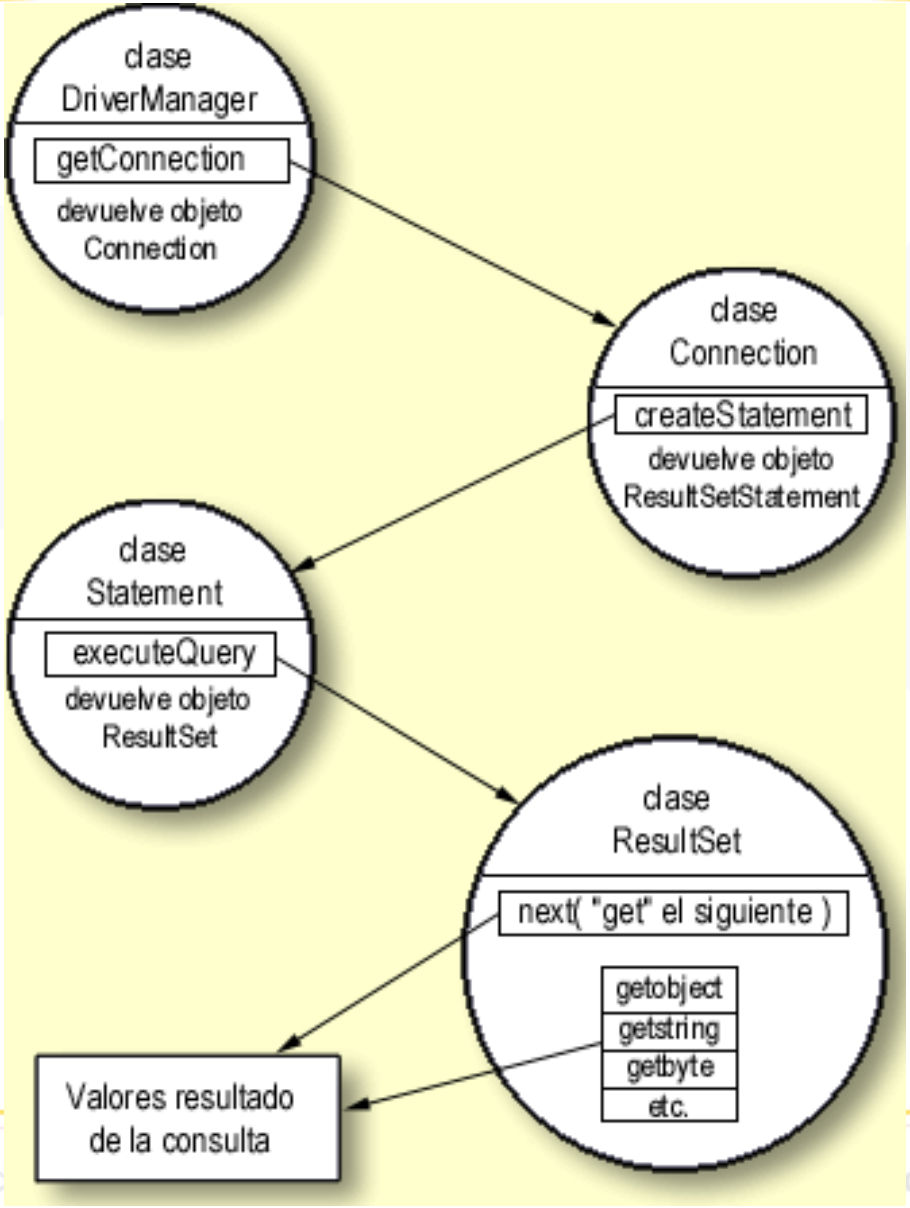
    p_sent.executeQuery(); //Retorna objeto de tipo ResultSet

Cerrar la Base de Datos

- Es importante cerrar/liberar las conexiones una vez que ya no son usadas. De esta manera el objeto **Connection** será marcado para ser recolectado por el Garbage Collector y, además teniendo en cuenta que la cantidad de conexiones disponibles a una BD es limitada, es importante cerrar las conexiones que no son más usadas.
- El método **destroy()** es el apropiado para llevar a cabo esta acción.

```
public void destroy() {
try {
    dbCon.close();
    } catch (Exception e) {.... }
}
```

Esquema de una aplicación con DB



Introduccion a SQL

Empleados				
empleado	apellido	nombre	categoria	departamento
00023	López	Alejandro	Ingeniero	022
00012	Gómez	Pedro	Ingeniero	011
00045	Pérez	Gonzalo	Perito	022

Departamentos		
departamento	jefe (Empleados)	sede
011	00023	Sevilla
022	00012	Madrid

SQL – Recuperar Información

- La sentencia SELECT es la que se utiliza cuando se quieren recuperar datos de la información almacenada en un conjunto de columnas. Las columnas pueden pertenecer a una o varias tablas y se puede indicar el criterio que deben seguir las filas de información que se extraigan.
- La sintáxis de la sentencia es:

```
SELECT <seleccion>
FROM <tablas>
WHERE <condiciones de seleccion>
[ORDER BY <columna> [ASC | DESC] [,<columna> [ASC |
    DESC]]...]
```

SQL – Recuperar Información

- Ejemplos:
  - SELECT * FROM Empleados WHERE departamento = '022';
  - SELECT * FROM Empleados WHERE departamento = '022' ORDER BY apellido;
  - SELECT apellido, nombre FROM Empleados WHERE departamento = '022' ORDER BY apellido;
  - SELECT CODIGO, APELLIDO, NOMBRE FROM Empleados AS e, Departamentos AS d WHERE d.sede = 'Sevilla' AND d.departamento = e.departamento;

SQL – Almacenar Información

- La sentencia INSERT se utiliza cuando se quieren insertar filas de información en las columnas.
- La sintáxis de la sentencia es:
  - INSERT INTO <nombre tabla>

  [(<nombre columna> [,<nombre columna]...)]

  VALUES (<expresion> [,<expresion>]...)
- Por ejemplo:
  - INSERT INTO Empleados VALUES (’00066’, ’Garrido’, ’Juan’, ’Ingeniero’ , ’022’);

SQL – Eliminar Información

- La sentencia DELETE es la que se emplea cuando se quieren eliminar filas de las columnas.
- La sintáxis es:
  - DELETE FROM <nombre tabla>

    WHERE <condicion busqueda>
- Si no se especifica la cláusula WHERE, se eliminará el contenido de la tabla completamente.
- Ejemplos:
  - DELETE FROM Empleados WHERE empleado=’00045’;
  - DELETE FROM Empleados;

SQL – Modificar Información

- Para actualizar filas ya existentes en las columnas, se utiliza la sentencia UPDATE.
- La sintáxis es:
  - UPDATE <nombre tabla>

    SET <nombre columna = (<expresion> | NULL) [, <nombre columna = (<expresion> | NULL)]...

    WHERE <condicion busqueda>
- Por ejemplo:
  - UPDATE Empleados

    SET nombre = 'Pedro Juan'

    WHERE empleado='00012';

Ver TestSql.java, Prueba.mdb, configurar el odbc.