Acceso a Datos

UT3. MANEJO DE CONECTORES
BASES DE DATOS RELACIONALES

1. Introducción

- Acceso a datos → Proceso de recuperación o manipulación de datos extraídos de un origen local o remoto.
- Los orígenes de datos pueden ser de muy diversa índole:
  - Una base de datos relacional
  - Una hoja de cálculo
  - Un fichero de texto
  - Un servicio web remoto
  - etc.
- En esta unidad vamos a centrarnos en los orígenes de datos relacionales.
- En esta unidad vamos a aprender a realizar aplicaciones Java para acceder a bases de datos relacionales partiendo del concepto y uso de los **conectores**.

2. El desfase objeto-relacional

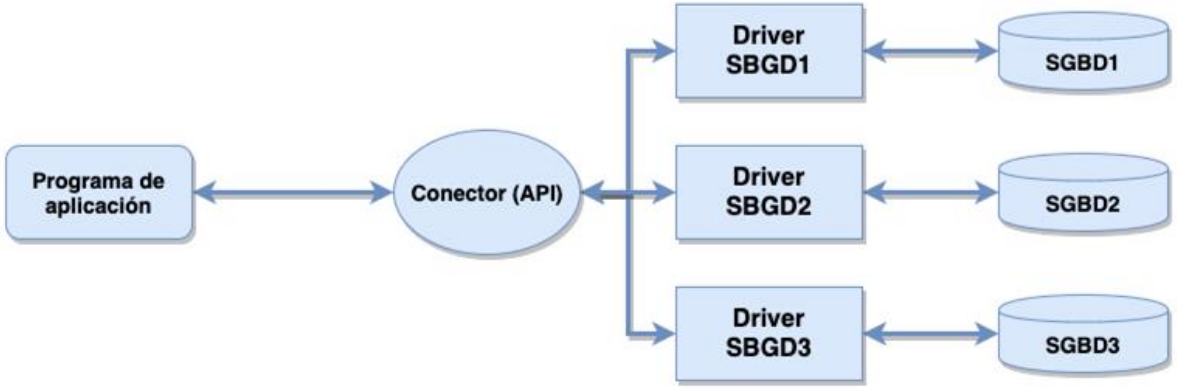
- A pesar de la existencia de bases de datos orientadas a objetos (OODB) el mercado está copado por las bases de datos relacionales pues ofrecen un rendimiento y flexibilidad superiores en diversos ámbitos.
- Sin embargo, el paradigma de programación dominante a día de hoy es la Programación Orientada a Objetos (POO).
- Las bases de datos relacionales no están diseñadas para almacenar objetos ya que existe un desfase entre las estructuras del modelo relacional y los modelos de datos utilizados en POO.
- Esto quiere decir que para aunar ambos conceptos se requiere de un esfuerzo extra a la hora de programar para adaptar unas estructuras a otras.
- A esto se le denomina el **desfase objeto-relacional** y se refiere a los problemas a los problemas que ocurren debido a las diferencias entre un modelo y otro.

3. Conectores

- Los sistemas gestores de bases de datos (SGBD) de distintos tipos tienen sus propios lenguajes especializados para operar con los datos que almacenan.
- Por su parte, los programas de aplicación se escriben en lenguajes de programación de propósito general, como Java.
- Para que estos programas puedan operar con los SGBD se necesitan mecanismos que permitan a los programas de aplicación comunicarse con las bases de datos en estos lenguajes.
- Estos se implementan en una API y se denominan **conectores**.

4. Conectores para RDBMS

- Para trabajar con sistemas gestores de bases de datos relacionales (RDBMS) se emplea el lenguaje SQL.
- Cada RDBMS utiliza su propia versión de SQL, con sus propias peculiaridades, es por ello que debemos usar sus propias estructuras de bajo nivel.
- El uso de drivers permite desarrollar una arquitectura genérica en la que el conector tiene una interfaz común tanto para las aplicaciones como para las bases de datos, y los drivers se encargan de las particularidades de las diferentes bases de datos.



4. Conectores para RDBMS

- De esta manera, el conector no es solo una API, sino una arquitectura, porque especifíca unas interfaces que los distintos drivers tienen que implementar para acceder a las bases de datos.
- Existen diferentes arquitecturas en este sentido:
  - ODBC
  - OLE-DB
  - ADO
  - JDBC
- Las predominantes a día de hoy son ODBC y JDBC ya que la casi totalidad de los SGBD las implementan y permiten trabajar con ellas.

- Los beneficios que proporcionan los conectores basados en drivers se consiguen a cambio de una mayor complejidad de programación.

4. Conectores para RDBMS

ODBC (Open Database Connectivity)

- Define una API que pueden usar las aplicaciones para abrir una conexión con la base de datos, enviar consultas, actualizaciones, etc.
- Las aplicaciones pueden usar esta API para conectarse a cualquier servidor de bases de datos compatible.
- Está escrito en lenguaje C.

JDBC (Java Database Connectivity)

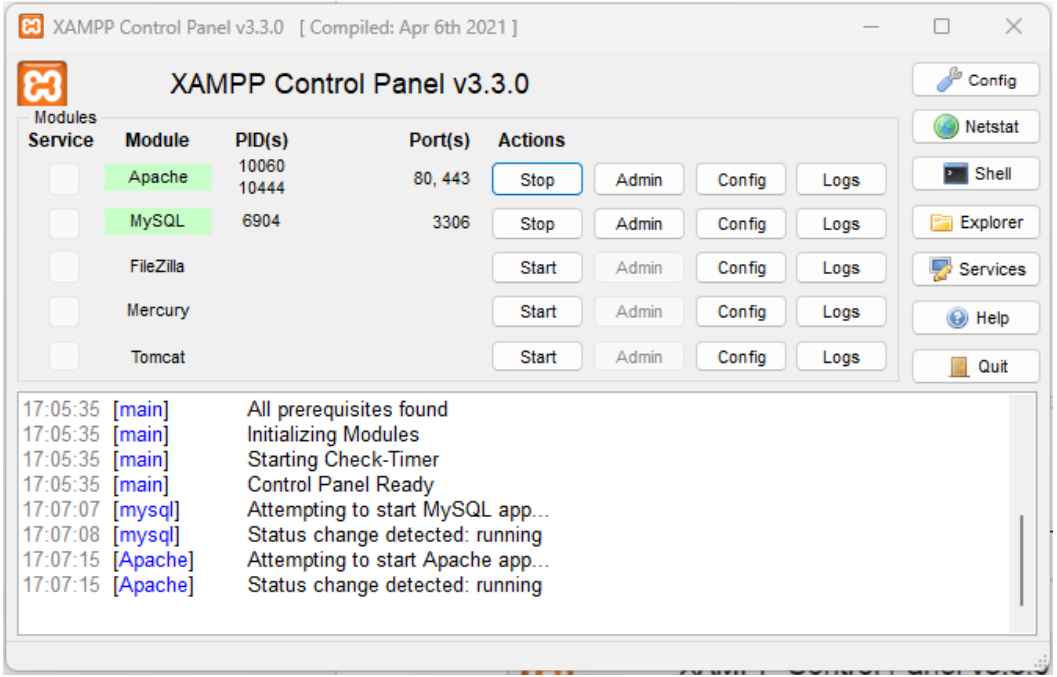
- Es la API por antonomasia de conexión de bases de datos con aplicaciones Java y es en la que nos centraremos en la presente unidad.

5. Acceso a DB mediante JDBC

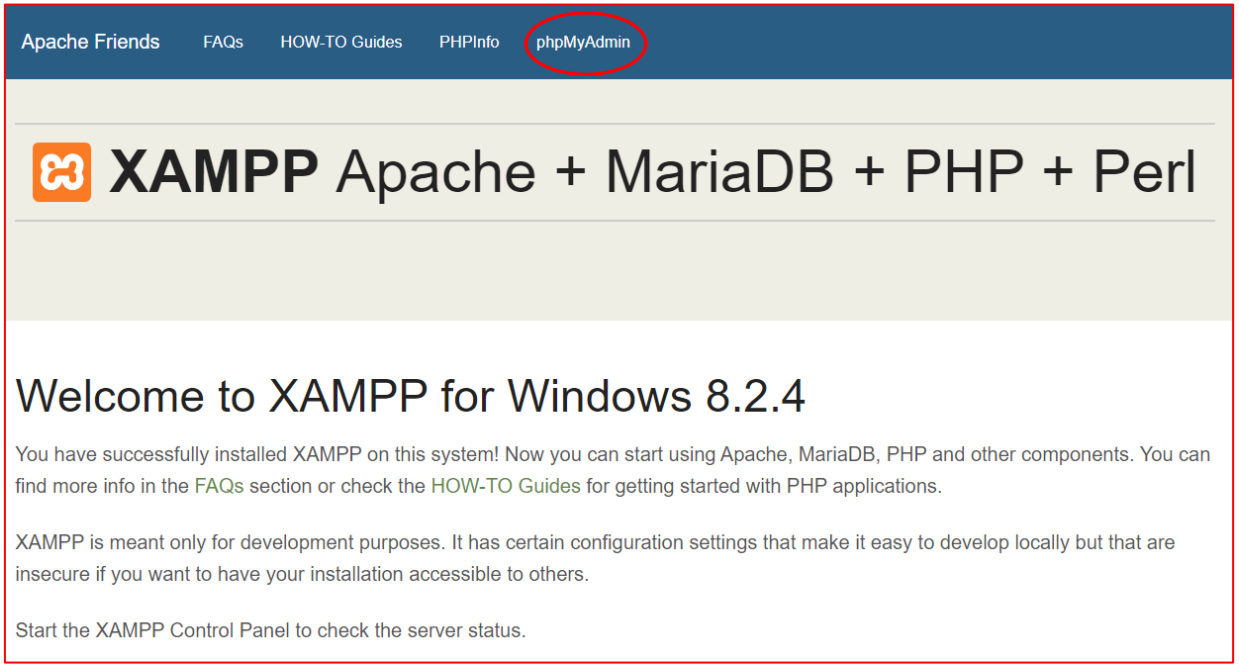
Conexión a MySQL:

- Para el siguiente ejemplo en vez de instalarnos solamente MySQL nos vamos a instalar el entorno de desarrollo **XAMPP** (**A**pache, **M**ySQL, **P**HP, **P**erl)
- https://www.apachefriends.org/es/index.html. Nos vamos a instalar la versión 8.2.4 (última a octubre de 2023), pero podríamos trabajar con otras versiones anteriores.
- Una vez instalado, arrancamos el servicio de MySQL y el de Apache (pulsando Start) y pulsamos en Admin para entrar en *phpMyAdmin*.
- Crearemos una base de datos, y la llamaremos **empresa.**

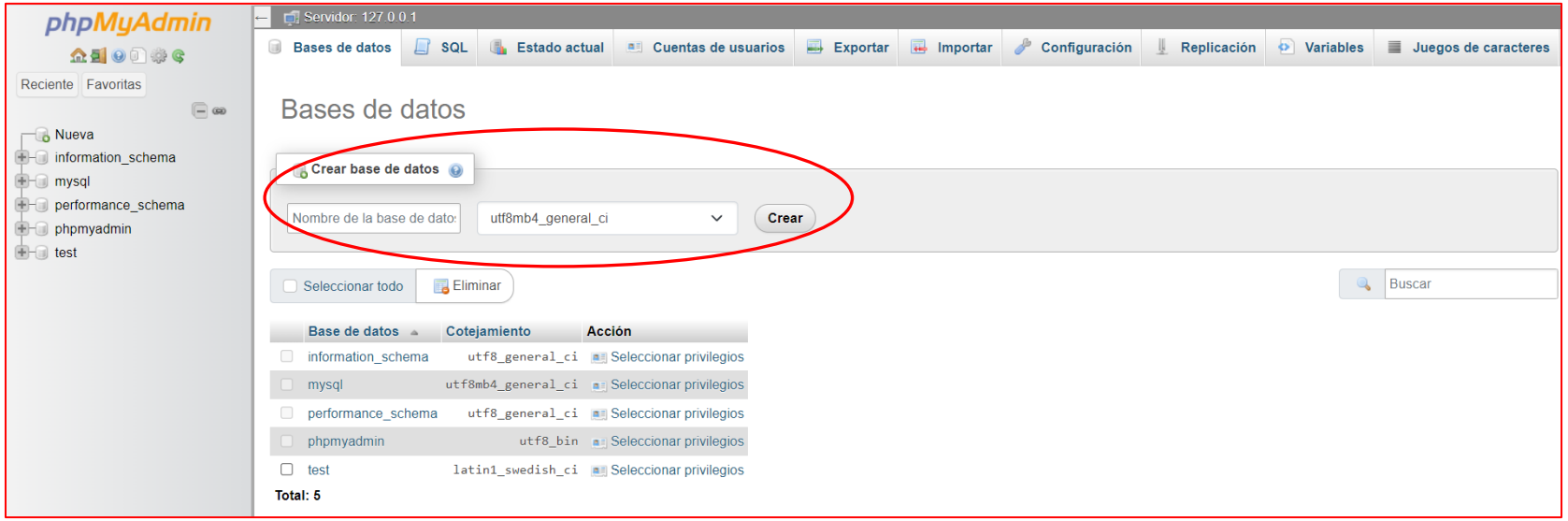
5. Acceso a DB mediante JDBC



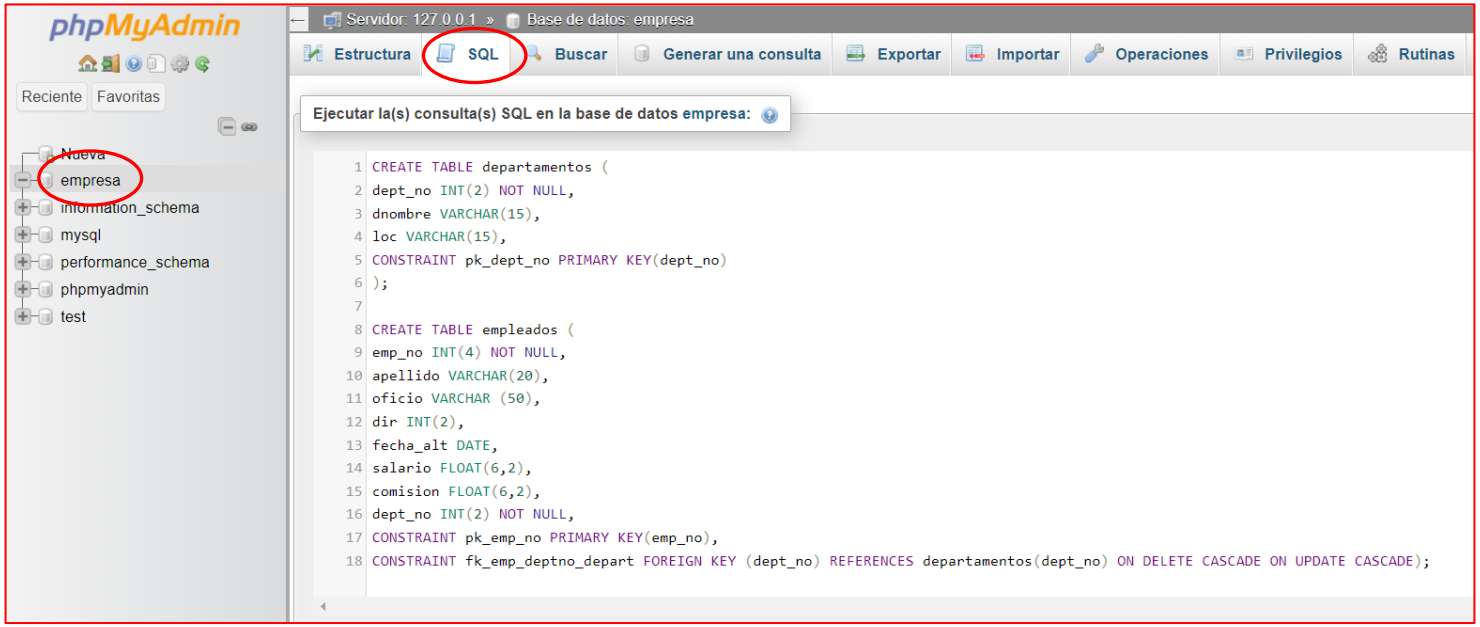
5. Acceso a DB mediante JDBC



5. Acceso a DB mediante JDBC



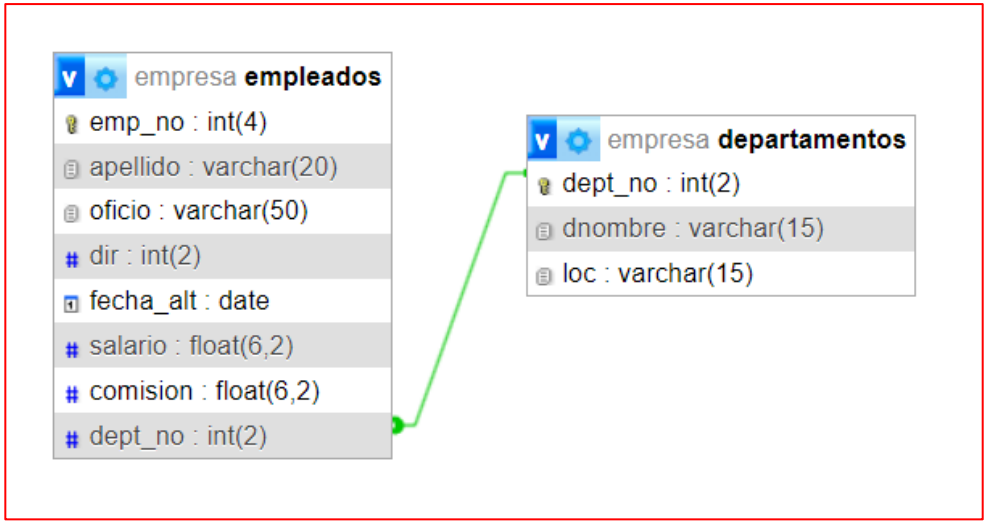
5. Acceso a DB mediante JDBC



5. Acceso a DB mediante JDBC

5. Acceso a DB mediante JDBC

- Opción **Más → Diseñador**:



5. Acceso a DB mediante JDBC

- Una vez creadas las tablas insertamos los siguientes valores en la tabla de departamentos:

```
INSERT INTO departamentos VALUES(10,'CONTABILIDAD','SEVILLA');
INSERT INTO departamentos VALUES(20,'INVESTIGACION','MADRID');
INSERT INTO departamentos VALUES(30,'VENTAS', 'BARCELONA');
INSERT INTO departamentos VALUES(40,'PRODUCCION','BILBAO’);
```

- También podríamos al dar de alta los empleados insertar la fecha actual valdría con pasar como parámetro la función sysdate():

```
INSERT INTO empleados (emp_no,apellido,oficio,fecha_alt,salario,comision,
dept_no) VALUES(10,'LOPEZ','PROGRAMADOR',sysdate(),1200,100,10);
```

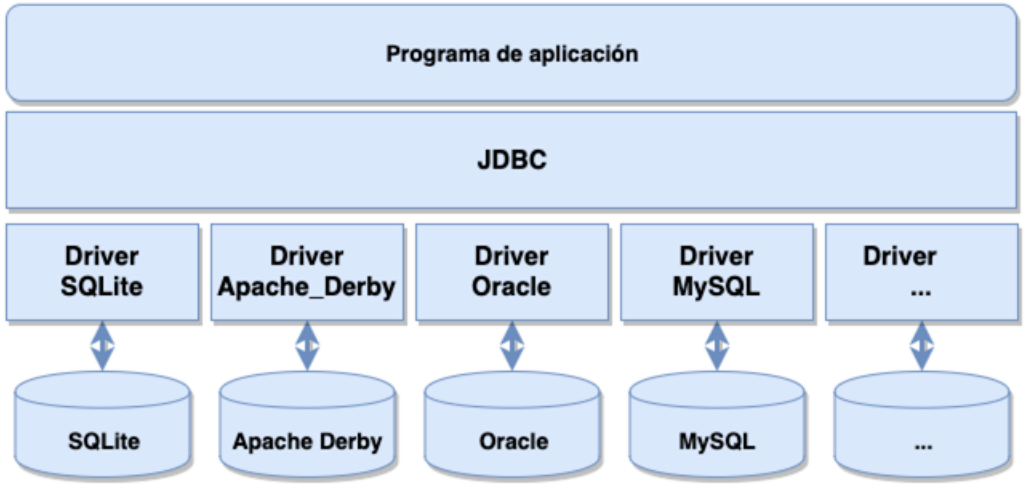
5. Acceso a DB mediante JDBC

- Para poder acceder a la base de datos desde eclipse hemos de obtener el JAR que contiene el driver MySQL (en el ejemplo se ha utilizado mysql-connector-java-8.0.30.jar) e incluirlo en el CLASSPATH o añadirlo a nuestro IDE, por ejemplo en Eclipse pulsamos en el proyecto con el botón derecho del ratón y seleccionamos **Build Paths → Add External Archives...** para localizar el fichero JAR.

- Desde la URL https://mvnrepository.com/artifact/mysql/mysql-connector-java elegimos la versión que deseamos descargar y nos descargamos el archivo .jar

5. Acceso a DB mediante JDBC

- JDBC proporciona una API para acceder a fuentes de datos orientados a bases de datos relacionales SQL.
- Provee una arquitectura que permite a los fabricantes puedan crear Drivers que permitan a las aplicaciones Java el acceso a datos.
- JDBC dispone de una interface distinta para cada SGBD, es lo que llamamos Driver.
- Esto permite que las llamadas a métodos Java se correspondan con la API de la base de datos.



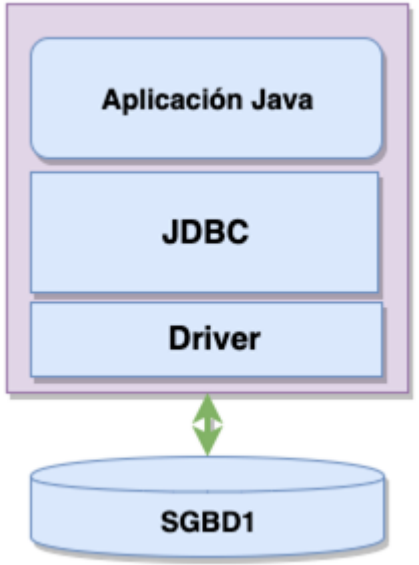
5. Acceso a DB mediante JDBC

- JDBC consta de un conjunto de interfaces y clases que nos permiten escribir aplicaciones en Java para gestionar las siguientes tareas con una base de datos relacional:
  - Conectarse a una base de datos.
  - Enviar consultas e instrucciones de actualización a una base de datos.
  - Recuperar y procesar los resultados recibidos de la base de datos en respuesta a dichas consultas.

5. Acceso a DB mediante JDBC

Modelo de acceso

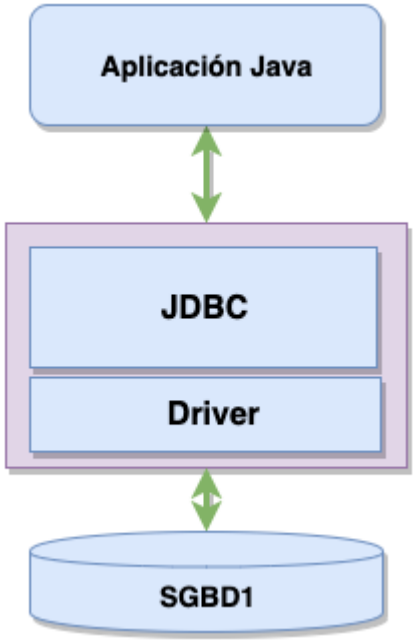
- La API JDBC es compatible con dos modelos de acceso:
  - **Modelo de dos capas** → La aplicación Java habla directamente con la base de datos, esto requiere un driver JDBC residiendo en el mismo espacio que la aplicación.
    - Desde el programa Java se envian sentencias SQL al SGBD para que las procese y este envía los resultados de vuelta al programa.
    - El SGBD puede encontrarse en la misma o en otra máquina y las peticiones se realizan a través de la red.
    - El driver es el encargado de gestionar las conexiones de forma transparente al programador.



5. Acceso a DB mediante JDBC

Modelo de acceso

- **Modelo de tres capas** → Los comandos se envían a una capa intermedia que se encarga de enviar los comandos SQL a la base de datos y de recoger los resultados.



5. Acceso a DB mediante JDBC

Tipos de drivers

a) **JDBC-ODBC Bridge →** Permite el acceso a SGBD JDBC mediante el protocolo ODBC.

b) **Native →** Controlador escrito parcialmente en Java y en código nativo de bases de datos. Traduce las llamadas de la API Java en llamadas propias del SGBD.

c) **Network →** Controlador de Java puro que utiliza un protocolo de red para comunicarse con el servidor de Base de datos.

d) **Thin →** Controlador de Java puro con protocolo nativo. Traduce las llamadas de la API a llamadas propias del protocolo de red utilizado por el SGBD.

5. Acceso a DB mediante JDBC

Funcionamiento JDBC

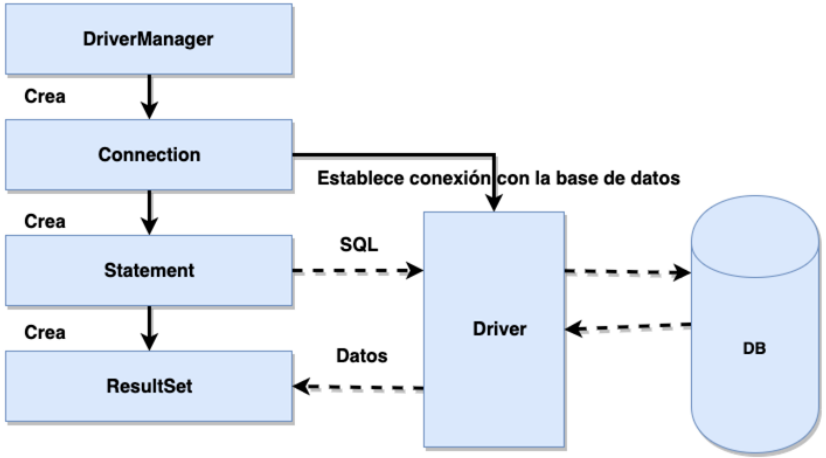
- JDBC define clases e interfaces en el paquete java.sql. La siguiente tabla muestra las más relevantes.

Clase o interface	Descripción
Driver	Permite conectarse a una base de datos
DriverManager	Permite gestionar los Driver instalados en el sistema
DriverPropertyInfo	Proporciona información del driver
Connection	Representa una conexión la base de datos
DatabaseMetadata	Proporciona información de la base de datos
Statement	Permite ejecutar sentencias SQL sin parámetros
PreparedStatement	Permite ejecutar sentencias SQL con parámetros
CallableStatement	Permite ejecutar sentencias SQL con parámetros de entrada y salida
ResultSet	Contiene el resultado de las consultas
ResultSetMetadata	Permite obtener información de un ResultSet

5. Acceso a DB mediante JDBC

Funcionamiento JDBC

- El funcionamiento de un programa con JDBC sigue los siguientes pasos:
  - Importar las clases necesarias
  - Cargar el Driver JDBC
  - Identificar el origen de datos
  - Crear un objeto Connection
  - Crear un objeto Statement
  - Ejecutar la consulta con el objeto Statement
  - Recuperar el resultado en un ResultSet
  - Liberar el objeto ResultSet
  - Liberar el objeto Statement
  - Liberar el objeto Connection



5. Acceso a DB mediante JDBC

Funcionamiento JDBC

Veamos un ejemplo de conexión y consulta a base de datos MySQL usando el siguiente esquema de bases de datos.

DBName → Empresa

TableName → Departamentos

- Campos:
  - dept_no INT PK
  - dnombre VARCHAR(15)
  - loc VARCHAR(15)

5. Acceso a DB mediante JDBC

Funcionamiento JDBC

En el código anterior hemos seguido los siguientes pasos

1. **Cargar el driver** → En primer lugar se carga el driver con el método forName de la clase Class, se le pasa un objeto String con el nombre de la clase del driver como argumento.

```
Class.forName("com.mysql.cj.jdbc.Driver");
```

1. **Establecer la conexión** → A continuación se establece la conexión con la BBDD usando la clase DriverManager y el método getConnection pasando como parámetros *URL, USER y PASS* de la BBDD.

```
Connection conexion=
        DriverManager.getConnection("jdbc:mysql://localhost/empresa", "root", "");
```

5. Acceso a DB mediante JDBC

Funcionamiento JDBC

3. **Ejecutar sentencias SQL** → A continuación realizamos la consulta a través de la interface Statement.
    i. Para obtener un objeto Statement se llama al método createStatement() de un objeto Connection válido.
    ii. El objeto Statement tiene el método executeQuery() que ejecuta una consulta en la BBDD
        1. Este método recibe como parámetro una String que debe contener una sentencia SQL.

```
Statement sentencia = conexion.createStatement();
String sql = "SELECT * FROM departamentos";
ResultSet resultado = sentencia.executeQuery(sql);
```

    i. El resultado se recoge en un objeto ResultSet, este objeto es una suerte de array o listado que incluye todos los datos devueltos por el SGBD

5. Acceso a DB mediante JDBC

Funcionamiento JDBC

4. **Recorrer el resultado** → Una vez hemos obtenido los resultados de la consulta y los hemos almacenado en nuestro ResultSet tan solo nos queda recorrerlos y tratarlos como estimemos oportuno.
    a. ResultSet tiene implementado un puntero que que apunta al primer elemento de la lista.
    b. Mediante el método next() el puntero avanza al siguiente elemento.
        i. El método next(), además de avanzar al siguiente registro, devuelve true en caso de existir más registros y false en caso de haber llegado al final.
        ii. Mediante los métodos getInt() y getString() vamos obteniendo los valores de las diferentes columnas. Estos métodos reciben el número de la columna como parámetro.

```
while (resultado.next()) {
    System.out.println(resultado.getInt(1)
            + " " + resultado.getString(2)
            + " " + resultado.getString(3));
}
```

5. Acceso a DB mediante JDBC

Funcionamiento JDBC

5. **Liberar recursos** → Por último debemos liberar todos los recursos utilizados para garantizar la correcta ejecución del programa.

```
conexion.close();
sentencia.close();
resultado.close();
```

5. Acceso a DB mediante JDBC

Funcionamiento JDBC

```
public static void selectMySQL() {
    try {
        // Cargar el driver
        Class.forName("com.mysql.cj.jdbc.Driver");

        // Establecemos la conexión con la BD
        Connection conexion =
                DriverManager.getConnection("jdbc:mysql://localhost/empresa", "root", "");

        // Preparamos la consulta
        Statement sentencia = conexion.createStatement();
        String sql = "SELECT * FROM departamentos";
        ResultSet resultado = sentencia.executeQuery(sql);

        // Recorremos el resultado para visualizar cada fila
        // Se hace un bucle mientras haya registros, se van visualizando
        while (resultado.next()) {
            System.out.println(resultado.getInt(1)
                    + " " + resultado.getString(2)
                    + " " + resultado.getString(3));
        }

        resultado.close();// Cerrar ResultSet
        sentencia.close();// Cerrar Statement
        conexion.close();// Cerrar conexión

    } catch (ClassNotFoundException en) {
        en.printStackTrace();
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Dudas y preguntas