índice

Tema 4 Manejo de conectores I.

Conectores para realizar conexión a base de datos desde Java.

Introducción al manejo de conectores

Conector o driver: serie de clases o librerías que se utilizan para conectar nuestra aplicación con la base de datos.

El desfase objeto-relacional

Discrepancia entre el aplicativo (programación orientada a objetos) y la base de datos relacional.

Base de datos relacionales se usan datos simples, mientras que en la orientada a objetos son objetos complejos.

Protocolos de acceso a base de datos

Un conector o driver: Es una serie de clases implementadas (API) que facilitan la conexión a la base de datos asociada.

En SLQ hay dos protocolos de conexión:

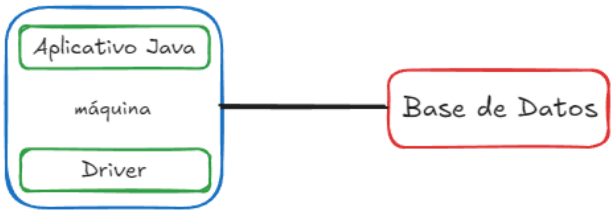
- JDBC (Java Database Connectivity): Desarrollado por Sun
- ODBC (Open DataBaseConnectivity): Desarrollado por Microsoft

Conexiones: Componentes y tipos

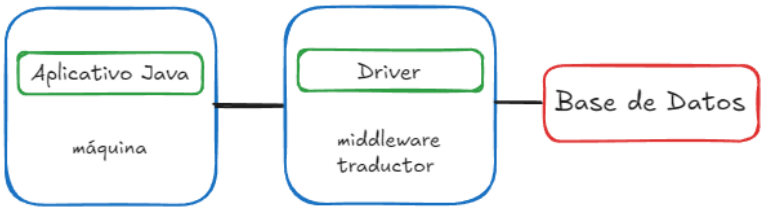
Conexión con el conector JDBC.

- **Componentes**
  - La API de JDBC: Tiene una serie de librerías y clases que nos facilitan el acceso a la base de datos relacionales. Paquetería: java.sql y javax.sql
  - Paquete de pruebas JDBC: estas clases se encargan de validar si un driver pasa los requisitos previstos por JDBC.
  - El gestor JDBC: encargado de realizar la unión entre nuestra aplicación Java con el driver. Hay dos formas de hacer la conexión.
    - Conexión directa
    - A través de un pool de conexiones.
  - El paquete JDBC-ODBC: facilita el uso de los drivers ODBC como si trabajásemos con JDBC.

- **Arquitectura en dos capas:** nuestra aplicación se conectará a la BDD a través de un driver. Ambos deben localizarse en el mismo sistema o máquina.



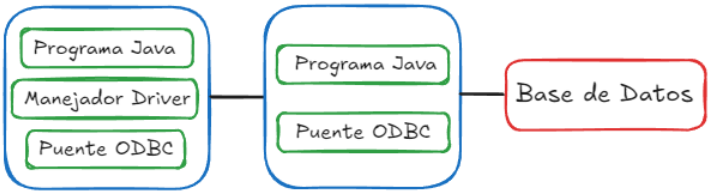
- **Arquitectura en tres capas:** nuestro aplicativo enviará las instrucciones a una capa intermedia (middleware). Esta capa cogerá la información y la envirá a la base de datos traduciendo los comandos que el aplicativo haya enviado.



Tipos

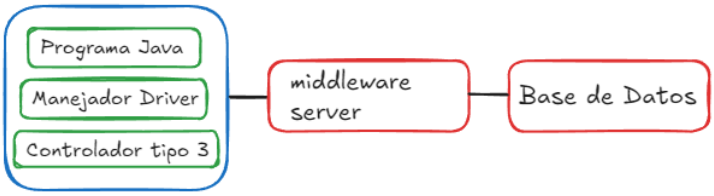
Distintos tipos de conexiones JDBC.

- **Driver tipo 1 JDBC-ODBC:** traduce las llamadas realizadas de JDBC a ODBC. Los datos devueltos se traducirán a JDBC.

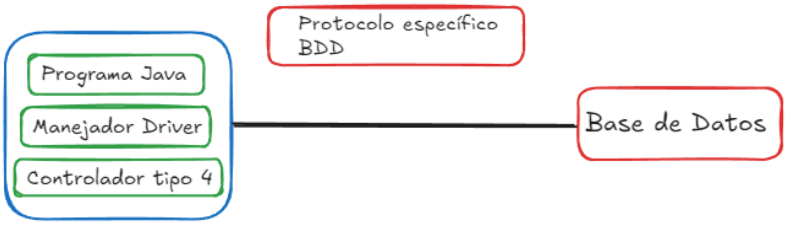


- **Driver tipo 2 JDBC Nativo:** estos drivers están escritos una parte en Java y otra parte, en código nativo. Las llamadas a la API JDBC son traducidas en llamadas propias de la base de datos relacional que tengamos.

- **Driver tipo 3 JDBC net:** es un driver de tres capas cuyas solicitudes JDBC están siendo traducidas en un protocolo de red en una capa intermedia. La capa intermedia recibirá las solicitudes y las enviará a la base de datos usando un driver JDBC de tipo 1 o tipo 2.



- **Driver tipo 4 protocolo nativo:** este tipo de driver realiza las llamadas mediante el servidor, usando el protocolo nativo del mismo. Estos drivers pueden desarrollarse en Java. Si se necesitara hacer un cambio de base de datos habría que desarrollar otro driver nativo adaptado a la nueva base de datos relacional.



Configuración de una conexión en código

1. Descargar el driver (suele ser ".jar")

2. Añadirlo a nuestro proyecto Java

3. Definir variables tipo String que nos van a servir para realizar la conexión con la base de datos.

4. Instanciamos una variable de tipo Connection y otra tipo Statement

```
// Variables de las credenciales de la BDD
final String usuario = "root";
final String password = "Med@c";
Connection dbConnection = null;
Statement statement = null;
```

- **Establecer conexión**

Podemos tener instaladas tantas conexiones como queramos.

El objetivo de la clase DriverManager es gestionar los drivers que tenemos en nuestra aplicación y permitir en una misma capa e acceso a todos.

Después de haber registrado el driver, se pueden usar los métodos estáticos para hacer getConnection.

Englobaremos todo en un bloque try/catch:

La primera instrucción que daremos es: "Class.forName()"; registrar el driver que antes hemos indicado en la variable estática "Driver".

El método getConnection le pasamos por pa´rametro la URL. Nos devolverá el método Connection llamado dbConnection.

- **Operaciones con variables y excepciones**

Una vez estemos conectados a la base de datos. Realizaremos una consulta simple y la almacenaremos en una variable de tipo String para luego ejecutarla.

Con la variable Connection ejecutamos el método createStatement y lo asignamos a la variable definida. Luego realizamos la consulta con el método executeQuery pasandole como parámetro la query definida en el String (la consulta).

El resultado de la query se asignará a una variable de tipo ResultSet y un while que por cada fila que nos devuelva la tabla podremos ir dando una vuelta más al bucle y seguir mostrando los resultados.

Y al final el catch con las excepciones.

SQL Exception: si a la hora de ejecutar el executeQuery algo va mal en base de datos ya sea gramaticalmente o sintácticamente

ClassNotFoundException: si no encuentra la librería, en la línea Class.forName(DRIVER).

Ventajas e inconvenientes del uso de conectores

Tipo de Driver	Ventajas	Inconvenientes
Tipo 1	Fácil acceso (incluidos en el paquete java)	Rendimiento bajo (demasiadas capas intermedias)
	Acceso a gran cantidad de drivers ODBC	Limitación de funcionalidad
		Problemas con applets en navegadores
Tipo 2	Rendimiento superior (llamadas nativas)	Librería de BDD debe iniciarse en el cliente
		No utilizables en Internet
	Interfaz nativa Java, aunque no es portable	

	entre plataformas	
Tipo 3	No necesita libería del fabricante en el cliente	Requiere código específico de BDD para la capa intermedia
	Mejor rendimiento en Internet, con opciones de portabilidad y escalabilidad	
Tipo 4	Buen rendimiento	Necesita un driver diferente para cada base de datos
	No requiere software especial en servidor o cliente	