1. Introducción a los conectores

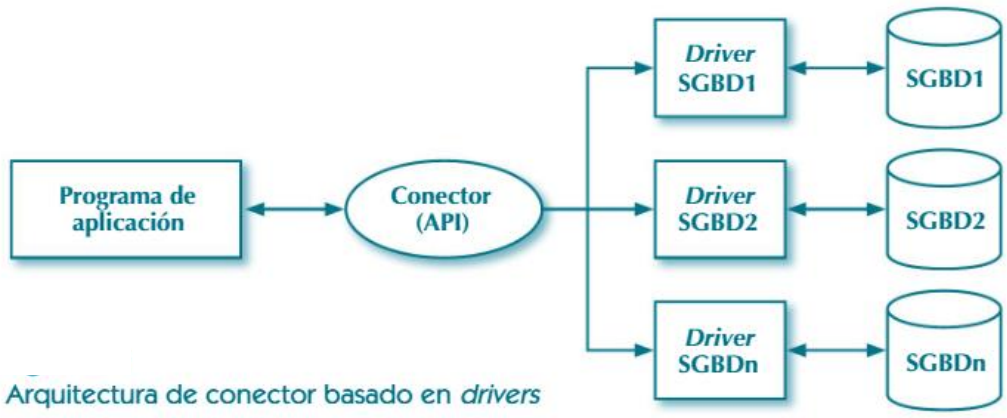
1.1. Conectores

Los sistemas gestores de bases de datos (SGBD) de distintos tipos (relacionales, de XML, de objetos o de otros tipos) tienen sus propios lenguajes especializados para operar con los datos que almacenan. En cambio, los programas de aplicación se escriben con lenguajes de programación de propósito general, como por ejemplo Java. Para que los programas de aplicación puedan interactuar con los SGBD, se necesitan mecanismos que permitan a los programas de aplicación comunicarse con las bases de datos en estos lenguajes. Estos mecanismos se implementan como una API y se denominan **conectores**.

1.2. Conectores para bases de datos relacionales

Los sistemas de bases de datos más utilizados hoy en día, con mucha diferencia, son los relacionales. Para trabajar con ellos se utiliza SQL. SQL es un lenguaje estándar, pero existen multitud de bases de datos relacionales distintas y cada una tiene su propia versión de SQL con sus propias particularidades. Además, cada base de datos tiene sus propias interfaces de bajo nivel.

Los conectores tienen una interfaz común tanto para las aplicaciones como para las distintas bases de datos. Entonces, ¿cómo se adaptan a las particularidades de cada base de datos? Utilizando ***drivers***.



Arquitectura de conector basado en *drivers*

Cada SGBD tiene su propio *driver*, que tiene que implementar la interfaz del conector para poder comunicarse con él. Por tanto, se puede decir que el conector no es solo una simple API, sino una arquitectura, ya que especifica unas interfaces a los diferentes *drivers*.

La primera arquitectura de conectores que surgió fue ODBC (Open Database Connectivity), desarrollada por Microsoft para Windows a principios de los noventa. ODBC es una API para el lenguaje C, y se usa hoy en día tanto en entornos Windows como Linux. A finales de los años noventa surgió JDBC como el equivalente a ODBC para Java, y es similar en muchos aspectos. De hecho, ambos están basados en el estándar X/Open SQL CLI, que especifica la manera en que un programa debe enviar sentencias de SQL a un SGBD y operar con *recordsets* (conjuntos de registros o filas) obtenidos.

Todas las bases de datos importantes proporcionan hoy en día *drivers* de ODBC y de JDBC. Existen *drivers* de JDBC para bases de datos relacionales, pero también para sistemas de almacenamiento basados en ficheros que almacenan datos de forma tabular o jerárquica (por ejemplo, ficheros CSV o XML).

El principal beneficio que nos proporcionan los conectores basados en *drivers* es su independencia de la base de datos. Esto se consigue a cambio de una mayor complejidad y, en algunos casos, de un menor rendimiento.

1.3. Acceso a resultados de consultas mediante conectores

Los conectores permiten realizar todo tipo de operaciones sobre una base de datos relacional. La cuestión fundamental que se plantea con los conectores es la correspondencia entre las estructuras de datos utilizadas en la base de datos y las estructuras de datos que utiliza el lenguaje de programación. En el caso de las bases de datos relacionales, la estructura de datos fundamental para el almacenamiento de la información es la tabla. Cada tabla tiene un conjunto fijo de columnas, cada una con un tipo de datos determinado.

Una consulta SQL devuelve un conjunto de filas de la tabla. Los conectores permiten recuperar estos resultados fila a fila, mediante un objeto que actúa como iterador o cursor. A continuación, se muestra la manera de realizar una consulta y obtener sus resultados utilizando conectores:

```
Connection c = getConnection(datos de conexión)
Statement s = c.createStatement();
ResultSet rs = s.executeQuery("SELECT... ");
while(rs.next()) { //En rs están disponibles más resultados de la consulta
    String dato1 = rs.getString(1); // obtener String de primera columna
    int dato2 = rs.getInt(2); // obtener int de segunda columna
}
s.close();
c.close();
```

Se ha utilizado la sintaxis del lenguaje Java y las clases de JDBC, pero en esencia es igual para cualquier base de datos relacional y para cualquier lenguaje de programación.

El objeto de tipo ResultSet actúa como iterador sobre los resultados de la consulta, que son un conjunto de filas. Una vez recuperada una fila, se puede acceder a cada dato indexando su posición o su nombre.

Este modelo de acceso es válido, con algunas diferencias, para otros tipos de bases de datos, tales como bases de datos de objetos y de XML. **Se trata siempre de abrir una conexión, realizar una consulta y utilizar un iterador o cursor para obtener uno a uno los resultados**. Según el tipo de base de datos, habrá diferentes operaciones para hacer avanzar el cursor, y se utilizarán diferentes estructuras de datos para recuperar los resultados individuales.

1.4. Desfase objeto-relacional

Hacer a la inversa el apartado anterior, es decir, almacenar los contenidos de variables del código en una base de datos relacional, puede ser más complicado. Especialmente cuando se trabaja con un lenguaje orientado a objetos y con objetos complejos (por ejemplo, objetos que contienen referencias a otros objetos). No es sencillo almacenar esta información en tablas y columnas. Al conjunto de dificultades que esto plantea se le conoce como desfase objeto-relacional, del inglés *object-relational impedance mismatch*.

Desfase objeto-relacional

Para la persistencia de objetos complejos hay dos posibilidades:

- Utilizar bases de datos orientadas a objetos, que permiten almacenar los objetos directamente.
- Utilizar técnicas o herramientas de mapeo objeto-relacional (ORM).

Esto se verá en unidades posteriores.

Ejercicio propuesto

Rellena la siguiente tabla, en la que debes indicar para cada base de datos la librería necesaria para su conexión mediante un programa Java y la URL desde donde la podemos descargar:

Base de datos	Librería Java	URL
SQLite	sqlite-jdbc-*version*.jar	https://github.com/xerial/sqlite-jdbc
MySQL		
MariaDB		
ORACLE		
Microsoft SQL Server		