1. Java Database Connectivity (JDBC)

1.1. Características de la API JDBC

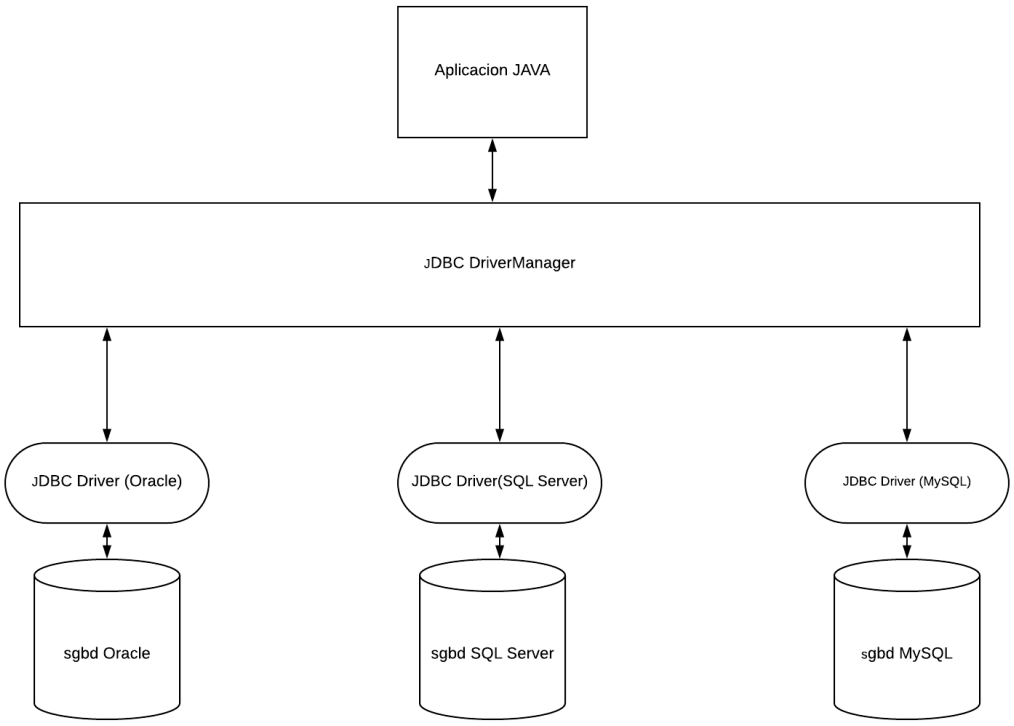
Java proporciona acceso a bases de datos relacionales empleando el conjunto de clases e interfaces de la API JDBC.

JDBC es una API bastante similar a ODBC en cuanto a funcionalidad, adaptada a las especificaciones de Java. Es decir, la funcionalidad se encuentra organizada en clases (porque Java es un lenguaje orientado a objetos) y, además, no depende de ninguna plataforma específica (se puede usar en Windows, Linux…). Entonces, el objetivo de JDBC es simplificar el acceso a bases de datos relacionales desde Java, permitiendo que las aplicaciones se puedan comunicar con los motores de bases de datos.

Esta API (**java.sql.***) fue desarrollada por Sun Microsystems con tres objetivos en mente:

- Ser una API con soporte de SQL, que permita construir sentencias SQL e insertarlas dentro de llamadas a la API de Java.
- Aprovechar la experiencia de las APIs de bases de datos existentes.
- Ser lo más sencilla posible.

Esta API está compuesta de varias capas, como se muestra en la siguiente figura:



La capa superior de este modelo es la aplicación Java. Una aplicación Java que utiliza JDBC puede comunicarse con muchas bases de datos diferentes, realizando pocas modificaciones (si es que las hay).

La clase **JDBC DriverManager** es responsable de localizar el driver JDBC que necesita la aplicación. Cuando una aplicación cliente solicita una conexión de base de datos, la solicitud se expresa en forma de URL. Una URL de JDBC es similar a las URL que se utilizan en los navegadores web. Un ejemplo de URL para Oracle podría ser: **`jdbc:oracle:thin:ejemplo/ejemplo@localhost:1521:XE`**

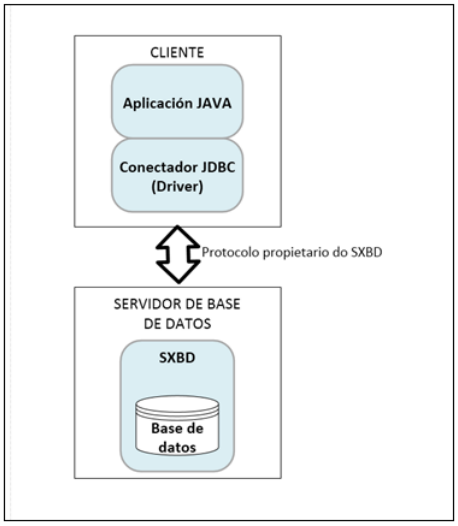
A medida que cada controlador se carga en una Máquina Virtual de Java (JVM), se registra con el JDBC DriverManager. Cuando una aplicación solicita una conexión, el DriverManager pregunta a cada controlador si puede conectarse a la base de datos especificada en la URL dada. Tan pronto como encuentra un controlador adecuado, la búsqueda se detiene y el controlador intenta establecer una conexión con la base de datos. Si el intento de conexión falla, el controlador lanzará una **SQLException** a la aplicación. Si la conexión se completa con éxito, el controlador crea un objeto de conexión y lo devuelve a la aplicación.

Hay que mencionar también que la API JDBC soporta dos modelos distintos de acceso a bases de datos:

Modelo de dos capas

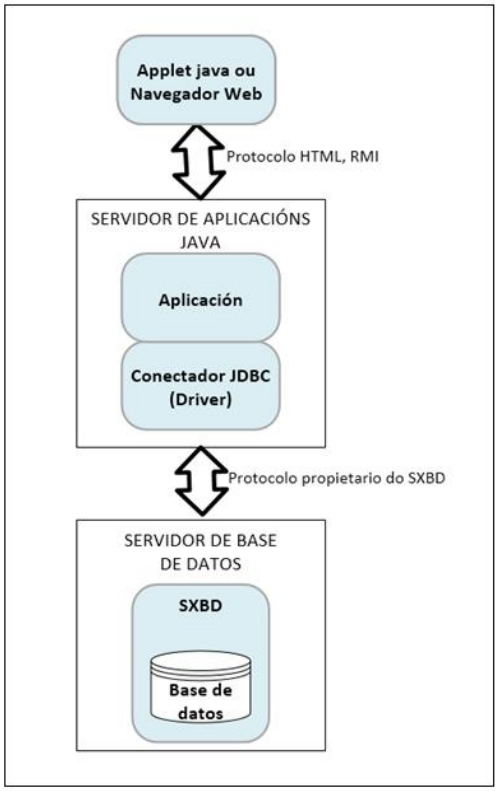
En este modelo la aplicación Java se conecta directamente al SGBD. Se necesita que el conector JDBC se almacene junto con la aplicación para que se pueda comunicar con el SGBD. Las sentencias SQL generadas en la aplicación cliente son enviadas a la base de datos y los resultados de la ejecución de estas sentencias se devuelven a la aplicación.

La aplicación cliente y la base de datos pueden estar en máquinas diferentes. Esta es la arquitectura típica de cliente-servidor, siendo el cliente la máquina del usuario que contiene la aplicación y siendo el servidor la máquina que contiene a la base de datos y el SGBD. La comunicación entre el cliente y el servidor se realiza a través de la red (Internet o Intranet).



Modelo de tres capas

En este modelo, las sentencias SQL son enviadas a una capa intermedia de servicios, que se encarga de enviarlas al SGBD. El SGBD procesa las sentencias y retorna los resultados a la capa intermedia, que se encarga de enviarlos a la aplicación del usuario.



La ventaja de este modelo es que el nivel intermedio mantiene en todo momento el control de las operaciones que se realizan en la base de datos. Además, los conectadores JDBC no tienen que residir en la máquina del cliente, lo que libera al usuario de la instalación de cualquier tipo de software adicional.

1.2. Conector JDBC

El conector de JDBC se encarga de implementar las interfaces de la API y acceder a la base de datos. Para poder conectarnos a una base de datos y lanzar consultas, necesitamos tener un driver adecuado. Normalmente, necesitaremos utilizar un fichero **.jar** que contiene una implementación de todas las interfaces de la API JDBC. El conector, normalmente, nos lo proporciona el propio fabricante de la base de datos.

Cuando se construye una aplicación que se conecta a una base de datos, JDBC oculta la implementación interna de la base de datos, de forma que el programador no tenga que preocuparse de esto y se centre en el código de su aplicación. Entonces, el código de nuestra aplicación debería ser el mismo, independientemente del driver que utilicemos.

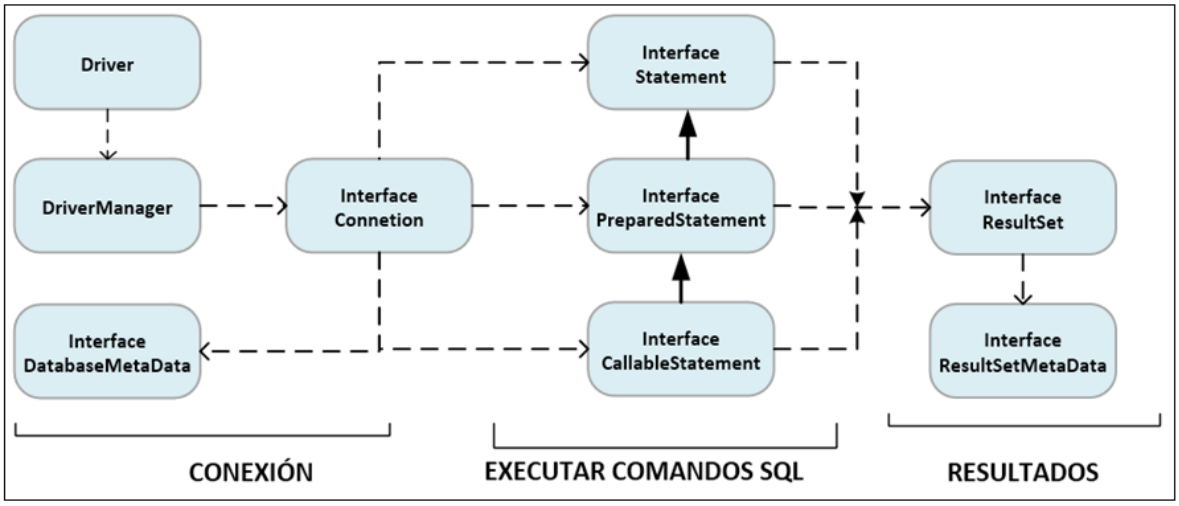
Para usar el conector JDBC, trabajaremos con el paquete **java.sql**.

El paquete java.sql contiene las clases e interfaces que componen la funcionalidad básica de JDBC:

- Conexión/desconexión de la base de datos.
- Envío de sentencias.
- Procesamiento de los resultados.
- Lectura de los metadatos sobre la estructura de la base de datos.

1.3. Clases e interfaces del paquete java.sql

Las clases e interfaces básicas de la API JDBC son las siguientes:



- La interfaz **Driver**: representa el punto de contacto entre una aplicación Java y el conectador que establece la conexión con la base de datos.
- La clase **DriverManager**: gestiona los conectadores de la base de datos y da soporte a la creación de nuevas conexiones. Se encarga de conectar la aplicación Java con el conectador JDBC correcto.
- La interfaz **Connection**: representa una sesión con una base de datos específica. Permite a los programadores crear objetos Statement que ejecutan sentencias SQL en la base de datos.
- La interfaz **DatabaseMetadata**: nos permite obtener, a través de Connection, información sobre la estructura de la base de datos y sobre las capacidades del conector JDBC.
- Las interfaces **Statement**, **PreparedStatement** y **CallableStatement**: la interfaz Statement funciona como un contenedor para ejecutar instrucciones SQL en una conexión dada. PreparedStatement permite almacenar instrucciones SQL precompiladas, de forma que puede ser ejecutada varias veces. CallableStatement permite ejecutar procedimientos.

- La interfaz **ResultSet**: representa un conjunto de datos devueltos al ejecutar una consulta SQL. Dentro de cada fila es posible acceder a los valores de sus columnas en cualquier orden.
- La interfaz **ResultSetMetadata**: permite obtener información sobre un objeto ResultSet.

1.4. Pasos a seguir para acceder a una BD relacional

Los pasos a seguir para acceder a una base de datos relacionar, utilizando JDBC, son:

1.- Instalar el conector de la base de datos.

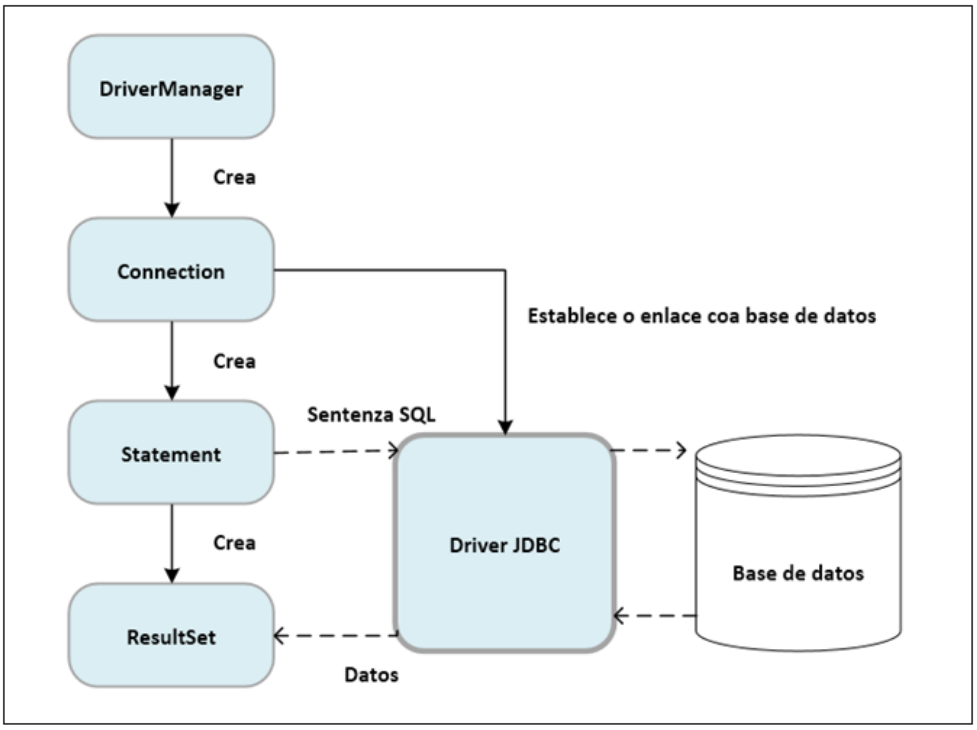
2.- Establecer una conexión con la base de datos.

3.- Crear y ejecutar sentencias SQL.

4.- Manejar los resultados de las sentencias.

5.- Liberar recursos y cerrar la conexión.

Representado en un esquema, sería algo como lo siguiente:



Paso 1: Instalar el conector

Debemos descargar el archivo **.jar** correspondiente y añadirlo en nuestro proyecto Java. Se debe añadir en nuestro buildpath con la opción de añadir un JAR externo.

Paso 2: Establecer una conexión

En Java, para establecer una conexión con una base de datos podemos utilizar el método **getConnection()** de la clase **DriverManager.** Este método recibe como parámetro la URL de JDBC, que identifica a la base de datos con la que queremos realizar la conexión. DriverManager busca entre los drivers registrados el que coincida con la URL. Si no lo encuentra, se lanza una SQLException. Si lo encuentra, devuelve un objeto **Connection** que representa a la conexión con la base de datos.

Un ejemplo en MySQL sería el siguiente código:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
public class PrimeraConexion {
     public static void main(String[] args) {
          try {
               // Establecemos la conexion con la BD
               Connection conexion = DriverManager.getConnection("jdbc:mysql://localhost/ejemplo", "root", "system");
               System.out.println("conecta");
               conexion.close(); // Cerrar conexión
          } catch (SQLException cn) {
               cn.printStackTrace();
          }
     }
}
```

La forma más sencilla de URL JDBC es una lista de tres valores separados mediante dos puntos. El primer valor de la lista representa el protocolo, que es siempre jdbc para los URL JDBC. El segundo valor es el nombre del SGBD (mysql, oracle, sqlserver…). El tercer valor es el nombre de sistema (servidor, puerto y nombre de la base de datos). Para MySQL la URL tiene el siguiente formato:

jdbc:mysql://[servidor]:[puerto]/[nombre_BD]

En nuestro caso, como MySQL escucha en el puerto 3306 y tenemos el servidor instalado en nuestra propia máquina, la URL será algo así:

jdbc:mysql://localhost:3306/[nombre_BD]

Paso 3: Añadir las sentencias SQL y manejar los resultados

La API JDBC distingue dos tipos de sentencias SQL:

- Consultas: SELECT.
- Modificaciones: INSERT, UPDATE, DELETE, sentencias DDL.

En primer lugar, vamos a ver cómo se realizaría una consulta. Tenemos que crear un objeto de tipo **Statement** y ejecutar la consulta con el método **executeQuery(sql)**. Este método devuelve un objeto **ResultSet**, que tendremos que recorrer para obtener los resultados. Básicamente, el objeto será el conjunto de filas que devuelve la consulta. Este objeto tiene un cursor que apunta a una de las filas devueltas (inicialmente se

coloca en la primera fila). Tenemos que ir avanzando por las filas utilizando el método **next()**. Podremos acceder a cada columna utilizando los métodos **get()** de la clase ResultSet.

Un ejemplo de consulta sería el siguiente:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
public class PrimeraConexion {
     public static void main(String[] args) {
          try {
               // Establecemos la conexion con la BD
               Connection conexion = DriverManager.getConnection("jdbc:mysql://localhost/ejemplo", "root", "system");
               Statement consulta = conexion.createStatement();
               ResultSet resultados = consulta.executeQuery("SELECT * FROM clientes");
               while (resultados.next()) {
                    System.out.println("El DNI es: " + resultados.getString("DNI"));
               }
               conexion.close();
          } catch (SQLException cn) {
               cn.printStackTrace();
          }
     }
}
```

Ahora veamos como ejecutar una sentencia de modificación. La estructura es la misma, pero utilizando el método **executeUpdate(sql)**. En estas sentencias no hay que recorrer un ResultSet, porque no devuelven filas. Un ejemplo sería:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
public class PrimeraConexion {
     public static void main(String[] args) {
          try {
               // Establecemos la conexion con la BD
               Connection conexion = DriverManager.getConnection("jdbc:mysql://localhost/ejemplo", "root", "system");
               Statement consulta = conexion.createStatement();
               int filasInsertadas =
                    consulta.executeUpdate("INSERT INTO clientes (DNI, NOMBRE, APELLIDOS) VALUES ('12345678K', 'Pedro', 'Lopez'");
               System.out.println("El numero de filas insertadas es: " + filasInsertadas);
               conexion.close();
          } catch (SQLException cn) {
               cn.printStackTrace();
          }
     }
}
```

También existe el método **execute(sql)**, que devuelve un boolean y sirve para cualquier sentencia. Devuelve true si es una consulta SELECT y false en caso contrario.

Paso 5: Cerrar la conexión

Simplemente tendremos que llamar al método **close()** de la clase **Connection**, como se hizo en los ejemplos.

EJERCICIO PROPUESTO

Utilizando la base de datos de *empleados*, crea una clase llamada *PrimeraConsulta* que ejecute la primera consulta realizada en los ejercicios que hicimos de SQL. Crea también una clase llamada *PrimeraModificacion*, que ejecute la primera modificación realizada en esos mismos ejercicios.

Puedes apoyarte en el código que se muestra en los apartados anteriores.