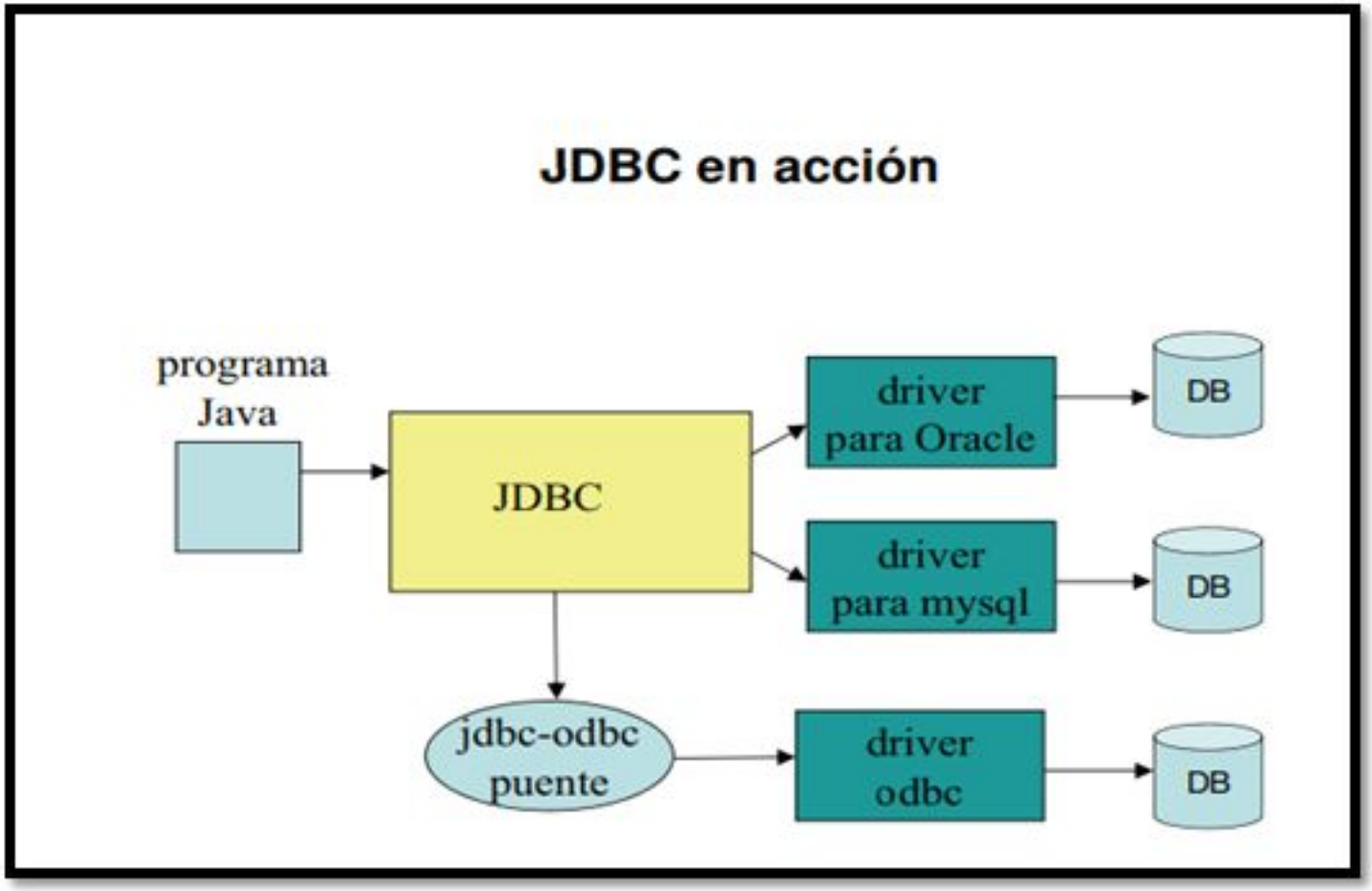
CONECTORES

JDBC

JDBC - JAVA

Java, mediante JDBC (Java Database Connectivity), permite simplificar el acceso a bases de datos relacionales, proporcionando un lenguaje mediante el cual las aplicaciones pueden comunicarse con motores de bases de datos. Sun desarrolló este API para el acceso a bases de datos, con tres objetivos principales en mente:

JDBC - JAVA


JDBC en acción
programa
Java
JDBC
driver
para Oracle
DB
driver
para mysql
DB
jdbc-odbc
puente
driver
odbc
DB

JDBC - JAVA

- **JDBC (Java Database Connectivity**) se trata de un API implementado específicamente para usar **con el lenguaje Java,** adaptado a las especificidades de Java.
- La funcionalidad se encuentra **encapsulada en clases** (ya que Java es un lenguaje totalmente orientado a objetos)
- **NO depende de ninguna plataforma específica**, de acuerdo con la característica multiplataforma defendida por Java.
- La idea en el desarrollo de JDBC era intentar ser tan sencillo como fuera posible, pero proporcionando a los desarrolladores la máxima flexibilidad.

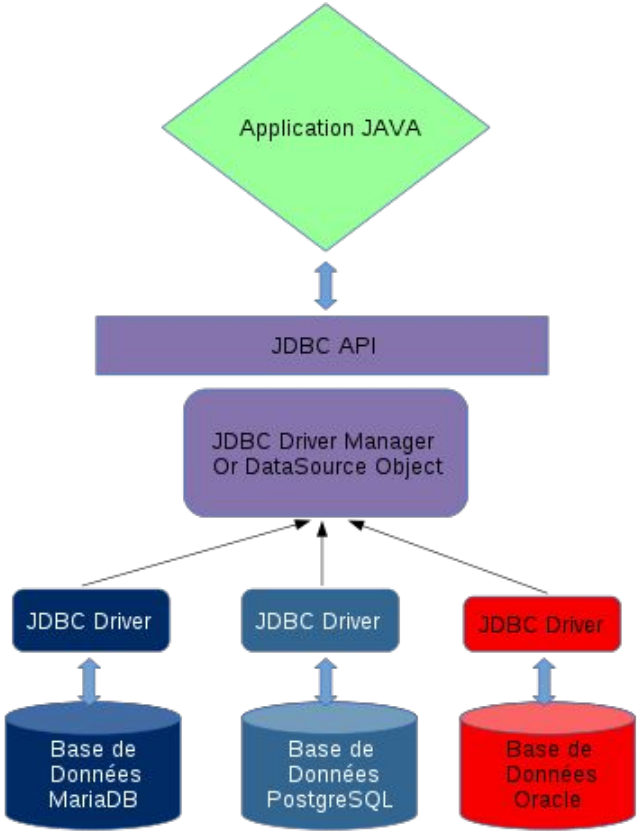
JDBC - JAVA

La aplicación que desarrollemos requerirá una **conexión a una base de datos** para poder persistir los datos que la aplicación gestione.

Java, mediante **JDBC,** permite simplificar el acceso a bases de datos relacionales, proporcionando un lenguaje mediante el cual las aplicaciones pueden comunicarse con motores de bases de datos.

El **desfase objeto-relacional** es aquel que surge cuando se desarrollan aplicaciones orientadas a objetos usando una base de datos de tipo relacional

JDBC - JAVA


Application JAVA
JDBC API
JDBC Driver Manager
Or DataSource Object
JDBC Driver
JDBC Driver
JDBC Driver
Base de
Données
MariaDB
Base de
Données
PostgreSQL
Base de
Données
Oracle

JDBC - JAVA

En una aplicación creada en Java, en muchos casos es necesario guardar datos en una base de datos. Para ello, es necesario establecer una conexión entre la aplicación y la base de datos.

Es necesario utilizar una interfaz para acceder a la base de datos desde Java con la API JDBC (**Java Database Conectivity**)

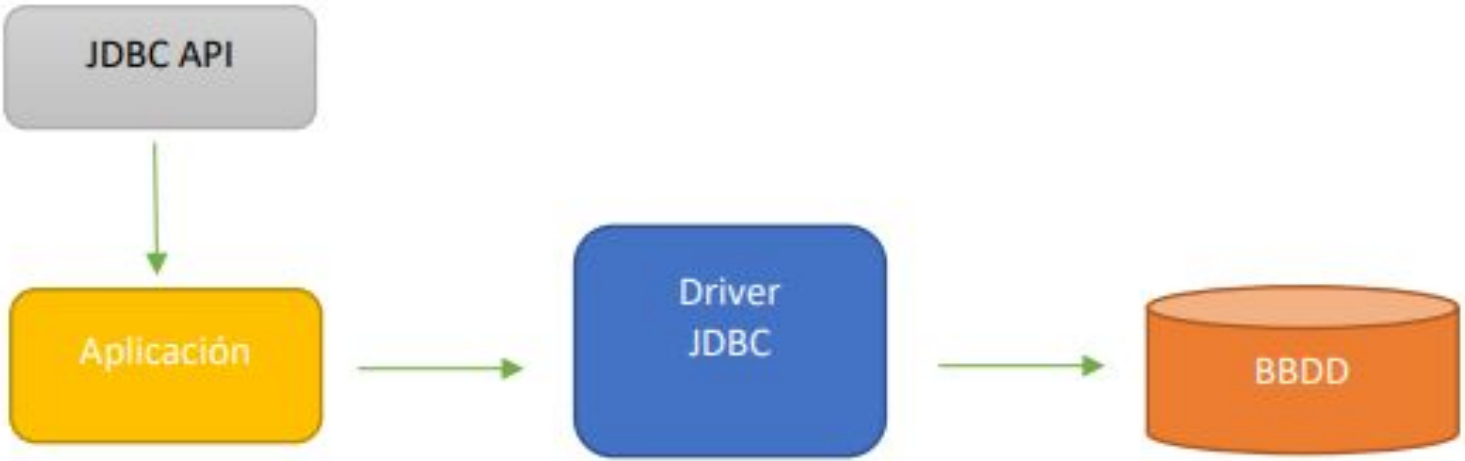
La clase **JDBC DriverManager** es responsable de localizar un controlador JDBC que necesita la aplicación. Cuando una aplicación cliente solicita una conexión de base de datos, la solicitud se expresa en forma de **una URL (Uniform Resource Locator).**

JDBC - JAVA

La API de JDBC soporta la comunicación entre la aplicación Java y **el driver que realizará la conexión entre la base de datos**, es decir, actúa como intermediaria. JDBC es la API común con la que interactúa el código de nuestra aplicación. Debajo está el controlador compatible con JDBC para la base de datos que vamos a utiliza.

Para realizar la conexión, necesitaremos **el driver JDBC**, que es el que nos permitirá que nuestra aplicación interactúe con la base de datos. Según qué base de datos queramos utilizar, tendremos que usar el driver correspondiente.

JDBC - JAVA



JDBC - JAVA

Registrar el driver JDBC

- Un **conector o driver** es un conjunto de clases encargadas de implementar los interfaces del API y acceder a la base de datos.
- Para poder conectarse a una base de datos y lanzar consultas, una aplicación n**ecesita tener un driver adecuado**.
- Un conector suele ser un fichero .jar que contiene una implementación de todas las interfaces del API JDBC.
- El conector lo proporciona el fabricante de la base de datos o bien un tercero.

JDBC - JAVA

- El código de nuestra aplicación trabaja mediante l**os paquetes java.sql y javax.sql.**
- JDBC ofrece las clases e interfaces para:
    1. **Establecer una conexión a una base de datos.**
    2. **Ejecutar una consulta.**
    3. **Procesar los resultados.**

JDBC - JAVA

Para registrar el driver usaremos **Class.forName().**

Al llamar al método **forName()** lo que hacemos es cargar de manera dinámica **el driver de la clase en memoria**. Necesitamos pasar por parámetro el driver de la base de datos que vamos a usar. Esta llamada solo se va a realizar una vez.

Si no usamos una base de datos MySQL, el procedimiento es el mismo, pero le pasaremos **el driver de la base de datos correspondiente.**

JDBC - JAVA

1. **Registrar el driver JDBC (base de datos oracle)**

```
try {
        Class.forName("oracle.jdbc.driver.OracleDriver");
    }catch(ClassNotFoundException ex) {
        System.out.println("Error al cargar el driver.");
    }
```

JDBC - JAVA

1. **Registrar el driver JDBC (base de datos mysql)**

```
try {
        Class.forName("com.mysql.jdbc.Driver");
    }catch(ClassNotFoundException ex) {
        System.out.println("Error al cargar el driver.");
    }
```

JDBC - JAVA

RDBMS	Nombre *driver* JDBC	Formato URL
MySQL	com.mysql.jdbc.Driver	jdbc:mysql://hostname/ databaseName
ORACLE	oracle.jdbc.driver.OracleDriver	jdbc:oracle:thin:@hostname:port Number:databaseName
DB2	COM.ibm.db2.jdbc.net.DB2Driver	jdbc:db2:hostname:port Number/databaseName

Tabla 11. Tabla esquemática según BBDD para la formación de la URL de conexión.

JDBC - JAVA

2. Establecer la conexión con la base de datos

Para ello, usaremos **DriverManager.getConnection().** Al llamar a getConnection() tendremos que pasarle una **URL** indicando qué **base de datos** usaremos, el **nombre, el usuario y la contraseña** para que pueda acceder a la BBDD. Una URL de JDBC es similar a las URL que utiliza con un navegador web.

String url = "jdbc:mysql://localhost/nombrebasedatos”;

String usuario= "root";

String contrasena = "password"

Connection conn = **DriverManager.getConnection(url, usuario, contrasena)**

JDBC - JAVA

EJEMPLO

```
public static Connection conectar_basedatos(String url,String usuario,String contrasena)
 {   Connection conexion;
    try{
       conexion = DriverManager.getConnection(url,usuario, contrasena);
        }catch(SQLException ex1)
        {    return null;
        }
       return conexion;
    }
}
```

JDBC - JAVA

Statement, PreparedStatement y CallableStatement

Se usa para enviar sentencias SQL a la base de datos.

Statement, PreparedStatement que hereda de Statement y CallableStatement que hereda de PreparedStatement.

Un objeto **Statement** se usa para ejecutar una sentencia SQL simple sin parámetros.

Un objeto **PreparedStatement** se usa para ejecutar sentencias SQL precompiladas con o sin parámetros

Un objeto **CallableStatement** se usa para ejecutar un procedimiento de base de datos almacenado**.**

JDBC - JAVA

La interface Statement nos suministra tres métodos diferentes para ejecutar sentencias SQL, **executeQuery, executeUpdate y execute**. El método a usar esta determinado por el producto de la sentencia SQL

El método **executeQuery** está diseñado para sentencias que producen como resultado un único result set tal como las sentencias SELECT.

El método **executeUpdate** se usa para ejecutar sentencias INSERT, UPDATE ó DELETE así como sentencias SQL DDL (Data Definition Language) como CREATE TABLE o DROP TABLE.

JDBC - JAVA STATEMENT

```
 Establecemos la conexión con la base de datos.
Connection conexion = DriverManager.getConnection ("jdbc:mysql://localhost/prueba","root",
"la_clave");

// Preparamos la consulta
Statement s = conexion.createStatement();
ResultSet rs = s.executeQuery ("select * from persona");

// Recorremos el resultado, mientras haya registros para leer, y
escribimos el resultado en pantalla.
while (rs.next())
{
    System.out.println (rs.getInt (1) + " " + rs.getString (2)+ " "
+ rs.getDate(3));
}

// Cerramos la conexion a la base de datos.
conexion.close();
```

JDBC - JAVA PrepareStatement

PreparedStatement psInsertar = conexion.prepareStatement("insert into person values (null,?,?,?)")

psInsertar.setInt(1, 23); // La edad, el primer interrogante, es un entero.

psInsertar.setString(2, "Pedro"); // El String nombre es el segundo interrogante

psInsertar.setString(3, "Perez"); // Y el tercer interrogante, un String apellido.

psInsertar.exequteUpdate(); // Se ejecuta la inserción.

psInsertar.setInt(1, 12); // La edad, el primer interrogante, es un entero.

psInsertar.setString(2, "Juan"); // El String nombre es el segundo interrogante

psInsertar.setString(3, "Lopez"); // Y el tercer interrogante, un String apellido.

psInsertar.exequteUpdate(); // Se ejecuta la inserción.

JDBC - JAVA ResultSet

ResutlIset

Un **ResultSet** es una clase de Java que sirve para almacenar datos de una consulta que hagamos con la clase Statement.

ResultSet lo podemos recorrer para mostrar los datos que nos interese.

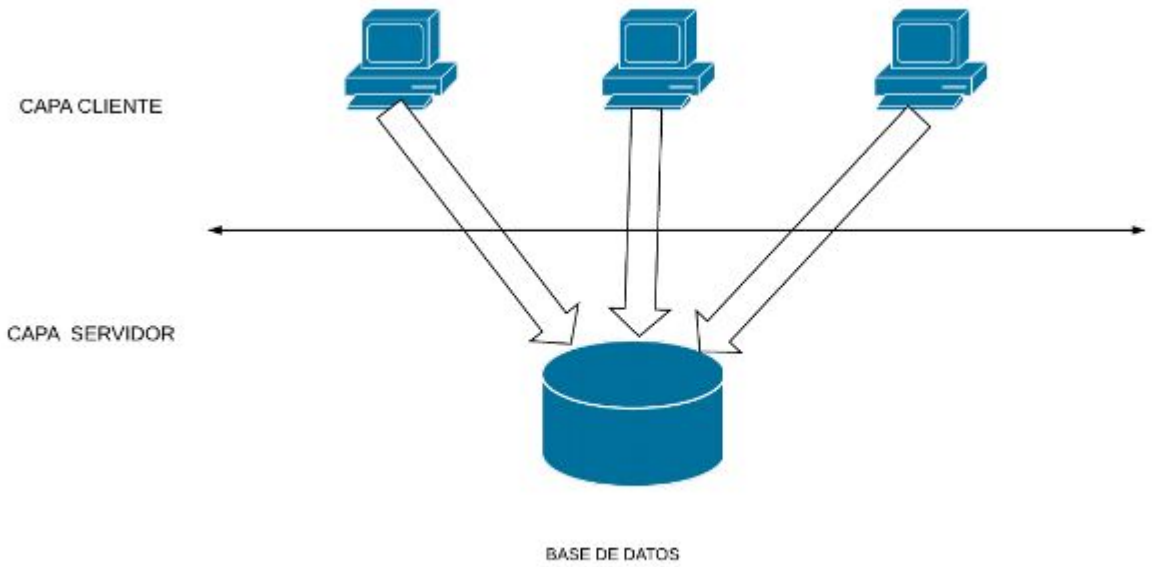
JDBC - JAVA

```
Statement statement = conexion.createStatement();
ResultSet resultSet = statement.executeQuery("select * from pok_pokemon");
System.out.println("NUM\tNOMBRE\t\tPESO\tALTURA\t");
 while (resultSet.next()) {
        System.out.print(resultSet.getInt("NUMERO_POKEDEX") + "\t");
        System.out.print(resultSet.getString("NOMBRE") + "\t");
        System.out.print(resultSet.getString("PESO") + "\t");
        System.out.print(resultSet.getString("ALTURA") + "\t");
       System.out.println("");
     }
```

JDBC - JAVA

```
// Se prepara el Statement
CallableStatement st = conexion.prepareCall( "{?=call una_funcion(?)}");
// Se indica que el primer interrogante es de salida.
st.registerOutParameter(1,Types.NUMERIC);
// Se pasa un parámetro en el segundo interrogante.
st.setInteger(2,parametro);
// Se hace la llamada a la función.
st.execute( );
// Se recoge el resultado del primer interrogante.
resultado = st.getInt( 1 );
```

ARQUITECTURA DE JAVA - Modelo de dos capas



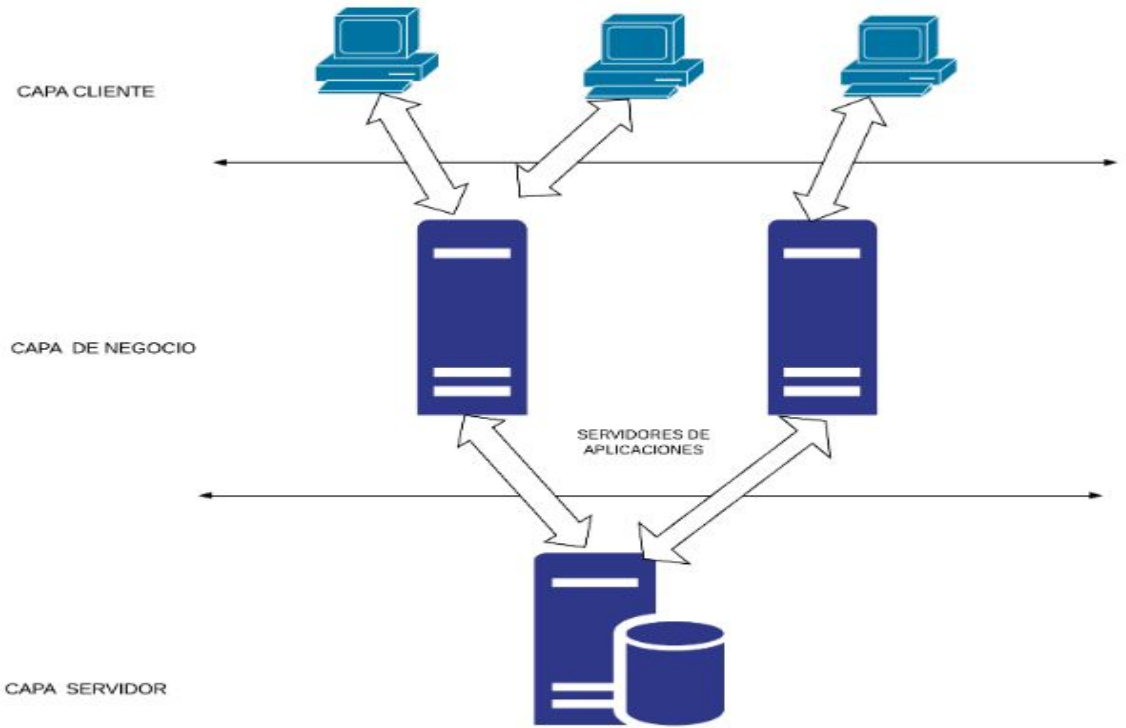
ARQUITECTURA DE JAVA - Modelo de dos capas

- En el modelo de dos capas, una aplicación se comunica directamente a la fuente de datos. Esto necesita un conector JDBC que pueda comunicar con la fuente de datos específica a la que acceder.
- Los comandos o instrucciones del usuario se envían a la base de datos y los resultados se devuelven al usuario. La fuente de datos puede estar ubicada en otra máquina a la que el usuario se conecte por red. A esto se denomina **configuración cliente/servidor**, con la máquina del usuario como cliente y la máquina que aloja los datos como servidor.

ARQUITECTURA DE JAVA - Modelo de tres capas

En el modelo de tres capas, los comandos se envían a una capa intermedia de servicios, la cual envía los comandos a la fuente de datos. La fuente de datos procesa los comandos y envía los resultados de vuelta la capa intermedia, desde la que luego se le envían al usuario.

ARQUITECTURA DE JAVA - Modelo de tres capas



ODBC

ODBC

¿Qué es el ODBC?

Open Data Base Conectivity

Si escribimos una aplicación para acceder a las tablas de una DB de Access, **¿Qué ocurrirá si después queremos que la misma aplicación, y sin reescribir nada, utilice tablas de SQL Server u otra DB cualquiera?** La respuesta es sencilla: no funcionará. Nuestra aplicación, diseñada para un motor concreto, no sabrá dialogar con el otro. Evidentemente, si todas las DB funcionaran igual, no tendríamos este problema.... aunque eso no es probable que ocurra nunca.

ODBC

- Un conector **ODBC (Open Database Connectivity)** es una interfaz estándar de programación que permite a las aplicaciones acceder y trabajar con bases de datos de una **manera uniforme y eficiente,** independientemente del sistema de gestión de bases de datos (DBMS) subyacente.
- Para utilizar ODBC, generalmente necesitas un controlador ODBC específico para la base de datos con la que deseas conectarte. Los controladores ODBC actúan como puentes entre la aplicación y la base de datos.
- Los desarrolladores configuran una fuente de datos ODBC (DSN) que contiene información sobre cómo conectarse a una base de datos específica, y luego la aplicación utiliza esta DSN para establecer la conexión.

ODBC

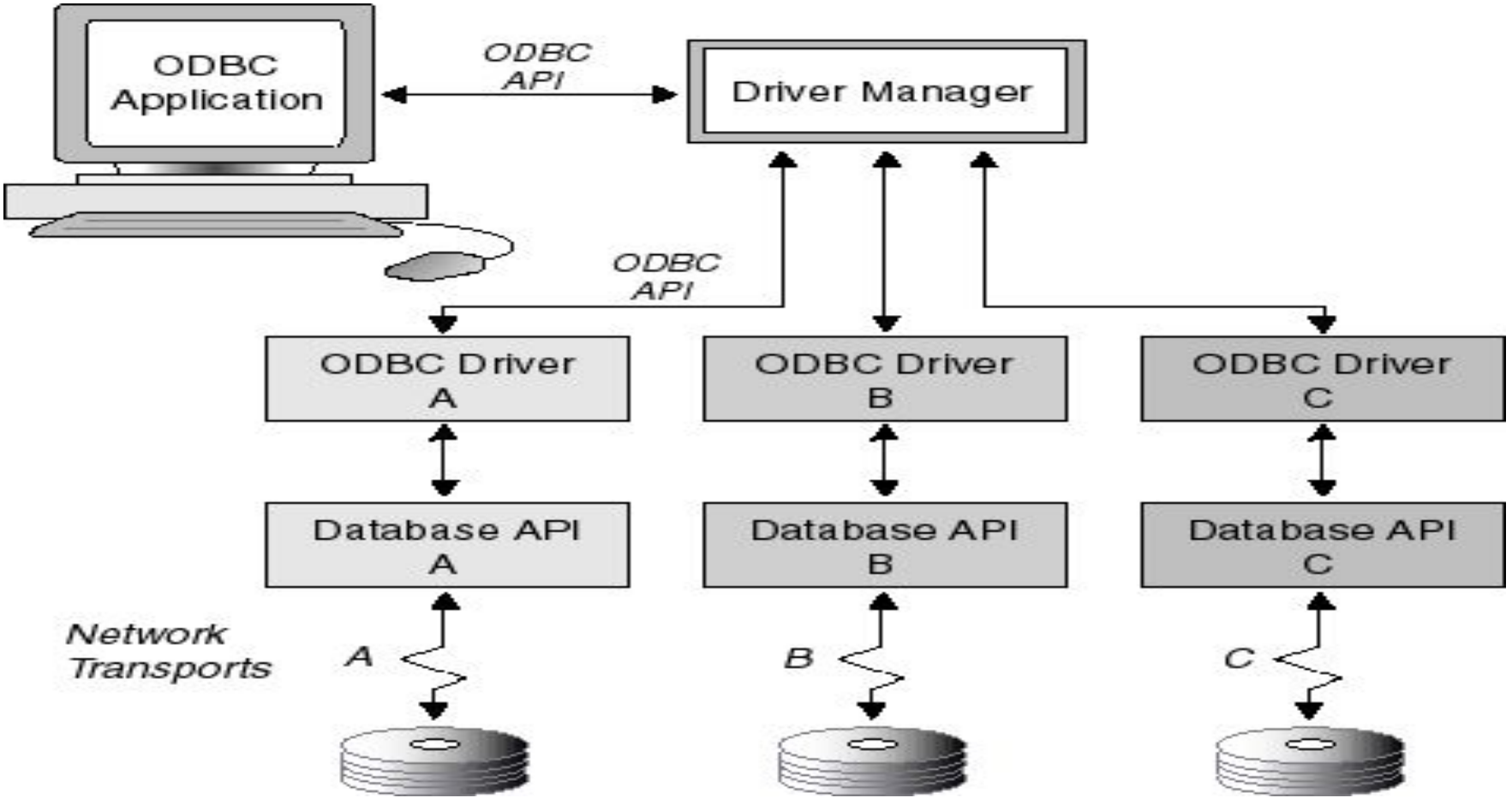
ODBC actúa como un **intermediario entre una aplicación y una base de datos**, permitiendo a las aplicaciones enviar consultas y recibir resultados sin preocuparse por los detalles específicos de cómo se almacenan o gestionan los datos en la base de datos subyacente.

Esto hace que sea más fácil para los desarrolladores crear aplicaciones que sean compatibles con una variedad de bases de datos, sin tener que reescribir código significativo cada vez que cambian la base de datos.

ODBC

Cualquier base de datos que se pretenda utilizar desde aplicaciones Windows debe tener su propio **driver ODBC.** Por ejemplo, MySQL dispone de un Driver ODBC que se puede descargar desde su página web. Las bases de datos Access (Microsoft Jet) y SQL Server de Microsoft también tienen su driver ODBC y este ya se encuentra instalado en el Windows de fábrica.

ODBC


ODBC Application
ODBC API
Driver Manager
ODBC API
ODBC Driver A
ODBC Driver B
ODBC Driver C
Database API A
Database API B
Database API C
Network Transports
A
B
C

ODBC - EJEMPLO

```
public class ODBCExample {
public static void main(String[] args) {
    // Nombre de la DSN configurada en tu sistema
     String dsnName = "mydsn";
 // URL de conexión JDBC usando el puente JDBC-ODBC
      String url = "jdbc:odbc:" + dsnName;
    ..........................................
```

ODBC - EJEMPLO

```
// Cargar el controlador JDBC-ODBC Bridge
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
//establecermos la conexión
connection = DriverManager.getConnection(url);
// Crear una declaración SQL
        statement = connection.createStatement();
        // Ejecutar una consulta SQL
        String sqlQuery = "SELECT * FROM nombre_de_tabla";
        resultSet = statement.executeQuery(sqlQuery);
```