ACCCESO A DATOS

PRUEBA ABIERTA UD2

ALUMNO CESUR

25/26

Alejandro Muñoz de la Sierra

PROFESOR

Patricio Santiago Fernández Florez

I N T R O D U C C I O N

El reto de la persistencia

La programación inicial suele centrarse en la lógica, los bucles y las clases. Pronto se hace evidente que una aplicación sin almacenamiento de datos tiene una utilidad limitada. La persistencia distingue un programa simple de uno funcional mediante la capacidad de gestionar y recuperar información.
La conexión del código Java con una base de datos requiere esfuerzo técnico. Existe un ecosistema variado con motores como **MySQL, Oracle y SQLite**, y cada uno utiliza su propio lenguaje. El objetivo de esta práctica fue comprender la conexión a estos sistemas de manera efectiva.

Esta memoria detalla una experiencia práctica más allá de las definiciones teóricas de **ODBC** y **JDBC**. Abordamos problemas reales en lugar de conceptos abstractos. Estos incluyen la gestión manual de librerías .jar, la resolución de conflictos de versiones en el JDK y la elección entre bases de datos de servidor o embebidas. Documentamos la conexión simultánea de una aplicación Java a un servidor **MySQL** y a una base de datos **SQLite**. También explicamos la solución a los errores de configuración del **Classpath** que surgieron durante el proceso.

0 1
A N Á L I S I S D E T E C N O L O G Í A S Y A R Q U I T E C T U R A S

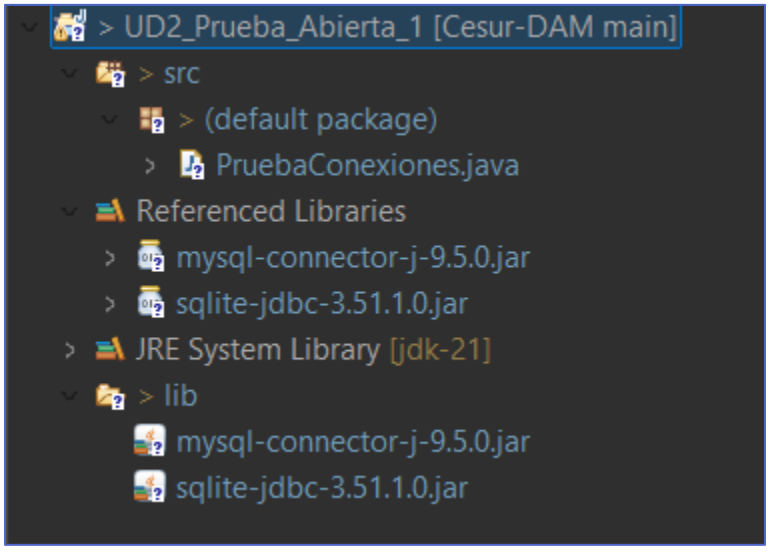
La aplicación Java requiere decisiones en dos niveles para guardar datos. Definimos el método de conexión y la arquitectura de almacenamiento.

1.1. Estándares: ¿Por qué JDBC y no ODBC?

Java necesita un intermediario para comunicarse con una base de datos. Existían dos opciones históricas, pero la práctica demostró la superioridad de una.

• **ODBC** (Open Database Connectivity): Microsoft creó este estándar antiguo. Funciona con muchos sistemas pero depende del Sistema Operativo. Utiliza **punteros** de **C** y exige la **configuración manual** de orígenes de datos en Windows. Esto afecta la portabilidad y la aplicación fallaría en Linux.

• **JDBC** (Java Database Connectivity): Esta es la solución nativa de Java y nuestra elección. Depende únicamente de la **Máquina Virtual (JVM)**. Un controlador funcional en Eclipse operará correctamente en otros servidores. La conexión es directa y orientada a objetos, lo cual simplifica la programación.
ODBC se utiliza ahora principalmente en sistemas heredados. JDBC es la opción lógica para nuevos desarrollos en Java debido a su rendimiento y facilidad de uso.



1.2. Arquitecturas: Servidor vs. Fichero

Establecimos la conexión con **JDBC**. Luego probamos dos tipos opuestos de bases de datos para observar sus diferencias:

Acceso.java | PruebaConexiones.java

```java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class PruebaConexiones {

    public static void main(String[] args) {
        System.out.println("--- INICIO DE PRUEBAS DE CONEXIÓN UD2 ---");

        // 1. PRUEBA CON SGBD INDEPENDIENTE (MySQL via XAMPP)
        // Aseguramos de tener XAMPP arrancado y Apache/MySQL en verde.
        probarMySQL();

        System.out.println("-----------------------------------------");

        // 2. PRUEBA CON SGBD EMBEBIDO (SQLite)
        // Con esto crearemos un fichero 'mi_base_embebida.db' en la raíz del proyecto.
        probarSQLite();

        System.out.println("--- FIN DE LAS PRUEBAS ---");
    }

```

1.**SGBD Independiente (MySQL)**: Este modelo Cliente-Servidor se usa comúnmente con XAMPP. La base de datos opera como un proceso independiente en el **puerto 3306**. Este sistema gestiona bien la concurrencia, la seguridad y los usuarios, características necesarias en una tienda en línea. Abrimos **XAMPP** y creamos la base de datos **Test** en **MySQL Workbench** para que se pueda generar la conexión.

```java
    private static void probarMySQL() {
        System.out.println("Intentando conectar a MySQL (Independiente)...");
        // Ajustamos usuario y contraseña

        String url = "jdbc:mysql://localhost:3306/test";
        String user = "root";
        String pass = "";

        try (Connection conn = DriverManager.getConnection(url, user, pass)) {
            if (conn != null) {
                System.out.println("☑ ÉXITO: Conectado a MySQL.");
                System.out.println("   Info del Driver: " + conn.getMetaData().getDriverName());
                System.out.println("   Ventaja: Ideal para sistemas multiusuario concurrentes.");
            }
        } catch (SQLException e) {
            System.err.println("✕ ERROR en MySQL: " + e.getMessage());
            System.out.println("   (Revisa que XAMPP esté encendido)");
        }
    }

```

2**. SGBD Embebido (SQLite)**: Este modelo no utiliza servidor. El motor es una **librería integrada** en la aplicación y la base de datos es un **archivo .db**. Aplicaciones móviles como WhatsApp usan esta arquitectura. La elegimos por su portabilidad, ya que copiar el archivo traslada la base de datos completa sin instalaciones adicionales.

```
43    private static void probarSQLite() {
44        System.out.println("Intentando conectar a SQLite (Embebido)...");
45        // La cadena de conexión es directa al fichero
46        String url = "jdbc:sqlite:mi_base_embebida.db";
47
48        try (Connection conn = DriverManager.getConnection(url)) {
49            if (conn != null) {
50                System.out.println("☑ ÉXITO: Conectado a SQLite.");
51                System.out.println("   Info del Driver: " + conn.getMetaData().getDriverName());
52                System.out.println("   Ventaja: Portabilidad total, la BD es un simple fichero.");
53            }
54        } catch (SQLException e) {
55            System.err.println("✕ ERROR en SQLite: " + e.getMessage());
56            System.out.println("   (Revisa que tengas el .jar de sqlite-jdbc en el Build Path)");
57        }
58    }
59 }
```

1.3. Retos Técnicos en la Implementación

Enfrentamos desafíos técnicos específicos. Evitamos gestores automáticos como Maven y configuramos las librerías manualmente para comprender mejor el proceso.

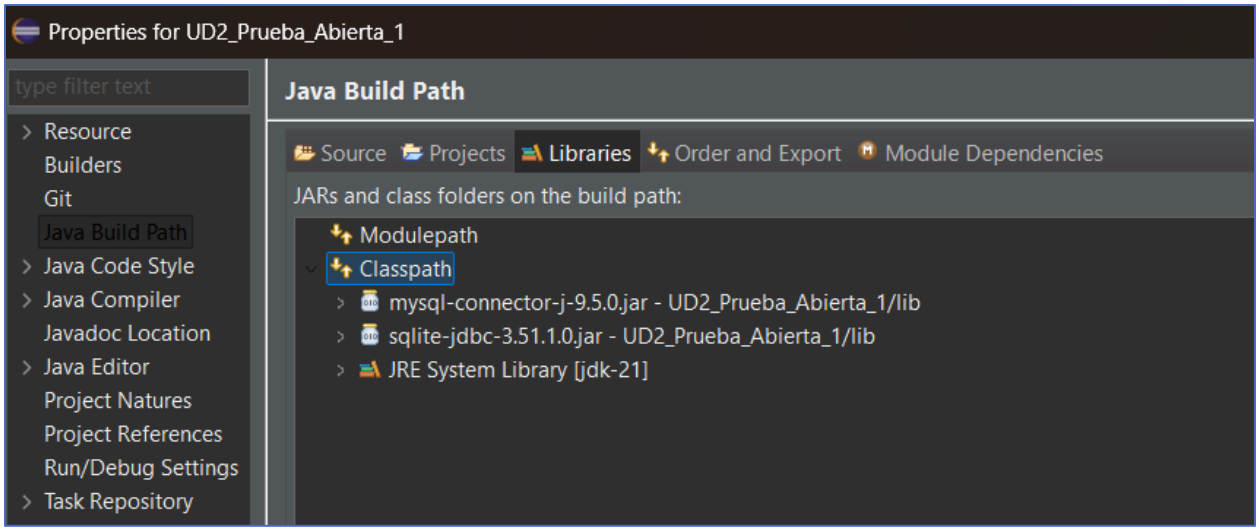
El conflicto del **Classpath**

Buscamos controladores específicos como **mysql-connector-j-9.5.0.jar** para MySQL y sqlite-jdbc para la base embebida. Aprendimos a través de errores que se deben usar rutas relativas dentro de la carpeta lib del proyecto al añadir JARs en Eclipse.

El error de versiones (UnsupportedClassVersionError)

Solucionar este fallo nos tomó mucho tiempo. Eclipse intentaba compilar con un **JDK moderno**, como la versión 25 o 22. La ejecución ocurría en la **versión LTS 21.** Esta diferencia causaba incompatibilidad en los archivos .class.

Forzamos el uso de la versión 21 en la configuración del compilador (Compiler Compliance Level) para solucionarlo. También movimos las librerías del Modulepath al Classpath clásico. El sistema modular de Java causa problemas de visibilidad con los drivers JDBC a veces.

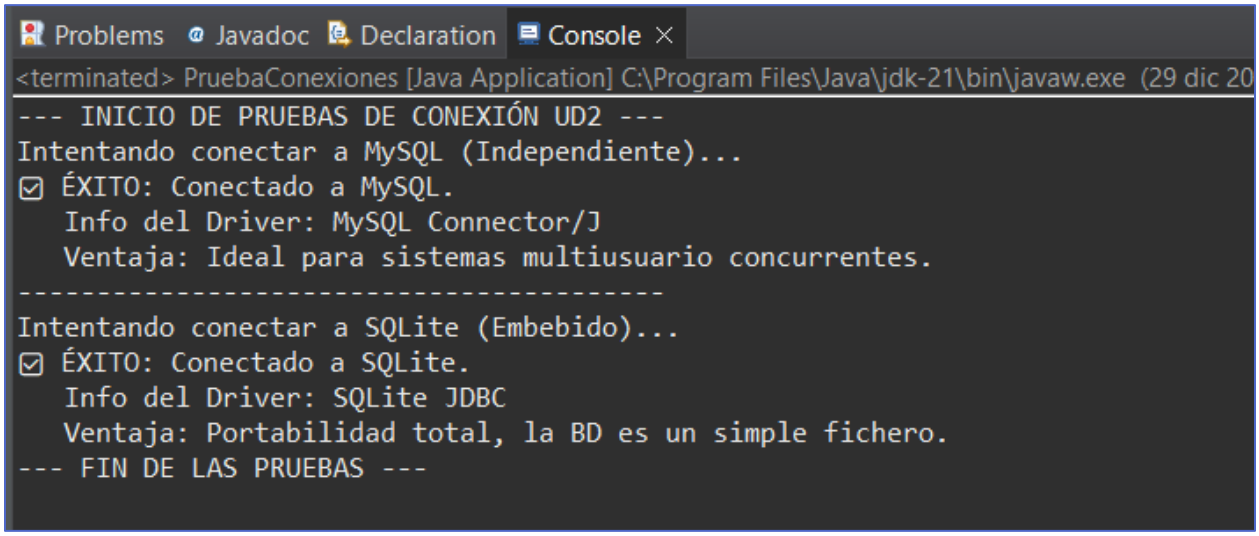


Resultado:

El código final crea dos **objetos Connection** distintos durante la misma ejecución:

1. jdbc:mysql://localhost:3306/test
2. jdbc:sqlite:mi_base.db

La consola de Eclipse muestra la conexión simultánea con MySQL y SQLite.



02
LOS DRIVERS JDBC: EVOLUCIÓN Y ELECCIÓN TÉCNICA

Investigamos durante la práctica. Vimos que los conectores varían. JDBC ha cambiado mucho desde los años 90. Revisamos los cuatro tipos existentes para explicar nuestra elección del driver:

Tipo 1 y 2 (Los antiguos): Estos dependían del sistema operativo o usaban puentes con ODBC. Eran lentos. La aplicación entera se cerraba tras un fallo en la librería nativa (.dll). Ya no se usan.

Tipo 3 (Red): Estos usan un servidor intermedio. Son útiles para bancos o sistemas complejos. Añaden demasiada dificultad para una aplicación estándar.

Tipo 4 (Puro Java): Este es el estándar actual. Nosotros usamos este tipo (mysql-connector y sqlite-jdbc).

¿Por qué el Tipo 4?
El driver está escrito 100% en **Java**. Se comunica directamente con el protocolo de la base de datos por la red ($Java \to Socket \to BD$). No requiere instalación en Windows ni en Linux. Cumple la promesa de "Write Once, Run Anywhere".

Tabla Comparativa Resumen

Tipo de Driver	Tecnología	Ventajas que hemos visto	Inconvenientes
Tipo 1 (Puente JDBC-ODBC)	Traduce llamadas Java a ODBC local.	Permitía acceso a BDs muy viejas (Access, FoxPro).	**Obsoleto.** Depende de tener configurado el ODBC en Windows. Es lento y ya no existe en Java 8+.
Tipo 2 (Nativo)	Java + Librerías C/C++ (DLLs).	Rendimiento decente en redes locales antiguas.	Muy peligroso: si falla la librería C, se cierra todo el programa Java. Requiere instalación en cada PC cliente.
Tipo 3 (Middleware)	Java -> Servidor Intermedio -> BD.	Muy seguro y flexible para arquitecturas corporativas complejas.	Añade latencia (un salto más en la red). Es complejo de montar para un proyecto normal.
Tipo 4 (Puro Java)	**100% Java directo al Socket.**	**El ganador.** Solo necesitas el .jar. Es rápido, portable y no requiere instalar nada extra en el cliente.	Cada base de datos necesita su propio .jar específico (el de MySQL es distinto al de SQLite).

El análisis muestra una conclusión clara. El **Tipo 4** es la única opción viable para el desarrollo profesional actual. Podemos encapsular la conexión en la carpeta **lib** o gestionarla con **Maven**. Esto facilita el despliegue.

0 3
C O N C L U S I O N E S

Esta práctica me ayudó a perder el miedo a la parte "invisible" de las bases de datos. Usamos herramientas visuales como Workbench a menudo. Olvidamos los procesos internos.
Destaco estos puntos:

1. La importancia de la configuración: Entender el Classpath y la ubicación de los .jar es necesario. Tuvimos problemas con las versiones del JDK. Aprendimos que la configuración del entorno importa tanto como el código.

2. El poder de JDBC: El código Java para conectar con MySQL y SQLite es casi idéntico. Solo cambian la URL y el driver. El método se aprende una vez. Funciona para cualquier base de datos.

Esto fundamenta el trabajo en la nube. Ahora apuntamos a localhost. En un trabajo real usaremos una IP en AWS o Azure. El driver y la lógica serán los mismos. Dominar este concepto es necesario ahora.

0 4
R E F E R E N C I A S

https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/

https://dev.mysql.com/doc/connector-j/en/

https://www.sqlite.org/whentouse.html

https://github.com/xerial/sqlite-jdbc

https://www.youtube.com/watch?v=d-_Z6qfoLOk

https://www.youtube.com/watch?v=sArp10JAIaM