Instituto Politécnico Nacional

Escuela Superior de Cómputo

Web Client and Backend Development Frameworks.

E1 : Aplicación con Acceso a Datos con JDBC

Profesor: M. en C. José Asunción Enríquez Zárate

Alumna: Mendez Cruz Ivonne

ivonne.mendezw@gmail.com

7CM1

March 11, 2025

Contents

List of Figures

1 Introducción

JDBC (Java Database Connectivity) es una API fundamental en el ecosistema de Java, diseñada para permitir la interacción entre aplicaciones Java y bases de datos relacionales. Como parte del estándar de Java, JDBC proporciona un conjunto de interfaces y clases que simplifican tareas como la conexión a bases de datos, la ejecución de consultas SQL y la manipulación de resultados. Esta API actúa como un puente entre el código Java y los sistemas de gestión de bases de datos (SGBD), permitiendo a los desarrolladores realizar operaciones complejas de manera eficiente y estandarizada.

En el contexto del desarrollo de software, la capacidad de interactuar con bases de datos es esencial para construir aplicaciones dinámicas y escalables. JDBC no solo facilita esta interacción, sino que también ofrece mecanismos para gestionar transacciones, controlar errores y optimizar el rendimiento de las consultas. Estas características hacen de JDBC una herramienta indispensable en el desarrollo de aplicaciones empresariales, sistemas de gestión de datos y cualquier software que requiera almacenamiento y recuperación de información estructurada.

En esta práctica, se implementó una aplicación que realiza operaciones básicas de CRUD (Crear, Leer, Actualizar y Eliminar) sobre una base de datos utilizando JDBC. El objetivo principal fue explorar y comprender los conceptos clave asociados con la conexión a bases de datos, la ejecución de consultas SQL y la gestión de transacciones. A través de esta implementación, se buscó no solo aplicar los fundamentos teóricos de JDBC, sino también identificar buenas prácticas para garantizar la eficiencia, la seguridad y la robustez en el manejo de datos.

2 Desarrollo

2.0.1 Preparación del Entorno

1. **JAR de JDBC para MySQL:**
   Es fundamental descargar el conector JDBC específico para MySQL, el cual permite establecer la comunicación entre la aplicación Java y la base de datos MySQL. Este archivo, generalmente denominado `mysql-connector-java-8.0.x.jar`, puede obtenerse directamente desde el sitio oficial de MySQL. El conector JDBC actúa como un puente que habilita las operaciones de base de datos desde el código Java.

2. **Instalación de MySQL:**
   Es necesario tener instalado y configurado un servidor de MySQL en el sistema donde se gestionará la base de datos. MySQL es un sistema de gestión de bases de datos relacionales (SGBD) ampliamente utilizado, y en este caso, almacenará las categorías y otros datos relevantes para la aplicación. Asegúrate de que el servidor esté en ejecución y accesible desde la aplicación Java.

3. **IntelliJ IDEA:**
   Como entorno de desarrollo integrado (IDE), se utilizará IntelliJ IDEA, una herramienta robusta y eficiente para el desarrollo de aplicaciones en Java. Se recomienda utilizar la última versión disponible para aprovechar las funcionalidades más recientes, mejoras de rendimiento y correcciones de errores. IntelliJ IDEA facilita la gestión de dependencias, la depuración de código y la integración con herramientas externas.

4. **Configuración inicial del proyecto:**
   Una vez descargado el archivo JAR del conector JDBC, es necesario agregarlo como dependencia en el proyecto de IntelliJ IDEA. Esto se realiza a través del panel de configuración del proyecto, en la sección de bibliotecas (*Libraries*). Asegúrate de que el archivo JAR esté correctamente vinculado para evitar errores de compilación o ejecución relacionados con la conexión a la base de datos.

2.0.2 Creación de la Base de Datos

Una vez configurado el entorno de desarrollo, el siguiente paso es crear la base de datos y las tablas necesarias para la aplicación. En este caso, se requiere una tabla para gestionar las categorías, la cual almacenará información como el identificador único, el nombre y la descripción de cada categoría.

A continuación, se presenta el script SQL utilizado para crear la tabla de categorías:

```
mysql> DESCRIBE ProgramaAcademico;
+--------------------+--------------+------+-----+---------+----------------+
| Field              | Type         | Null | Key | Default | Extra          |
+--------------------+--------------+------+-----+---------+----------------+
| idProgramaAcademico | int          | NO   | PRI | NULL    | auto_increment |
| nombre             | varchar(100) | NO   |     | NULL    |                |
| descripcion        | text         | YES  |     | NULL    |                |
| fecha              | date         | YES  |     | NULL    |                |
+--------------------+--------------+------+-----+---------+----------------+
4 rows in set (0.05 sec)
```

Figure 1: Base de Datos de ProgramaAcademico

2.0.3 Implementación del Codigo

El código de la aplicación se organiza en varias clases, cada una con una función específica. A continuación, se describen las principales clases utilizadas en el proyecto

La clase Main es el punto de entrada de la aplicación. En esta clase se inicializa la interfaz gráfica de usuario y se ejecuta la aplicación

```
public static void main(String args[]) {

    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new ProgramaAcad().setVisible(true);
        }
    });
}
```

Figure 2: Main de Programa Académico

La clase ConexionDB.java maneja la conexión a la base de datos. Su implementación es crucial para realizar operaciones de consulta y modificación de los datos en la tabla de categorías

```
public class ProgramaAcademicoApp {
    private static final String SQL_INSERT = "INSERT INTO ProgramaAcademico (nombre, descripcion, fecha) VALUES (?, ?, ?)";
    private static final String SQL_UPDATE = "UPDATE ProgramaAcademico SET nombre=?, descripcion=?, fecha=? WHERE idProgramaAcademico=?";
    private static final String SQL_DELETE = "DELETE FROM ProgramaAcademico WHERE idProgramaAcademico=?";
    private static final String SQL_SELECT = "SELECT * FROM ProgramaAcademico WHERE idProgramaAcademico=?";
    private static final String SQL_SELECT_ALL = "SELECT * FROM ProgramaAcademico";

    private Connection obtenerConexion() throws SQLException {
        String url = "jdbc:mysql://localhost:3306/7CM1?useSSL=false&allowPublicKeyRetrieval=true&serverTimezone=UTC";
        String usuario = "root";
        String clave = "chorizo12";

        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            return DriverManager.getConnection(url, usuario, clave);
        } catch (ClassNotFoundException | SQLException e) {
            throw new SQLException("Error en la conexión con la base de datos.", e);
        }
```

Figure 3: Conexion a la Base de Datos

La clase ProgramaAcdemico.java representa la entidad categoría, encapsulando los atributos y métodos que la describen dentro del sistema.

```java
public class ProgramaAcademico implements Serializable {
    private Long idProgramaAcademico;
    private String nombre;
    private String descripcion;
    private Date fecha;

    public ProgramaAcademico() {}

    public ProgramaAcademico(Long id, String nombre, String descripcion, Date fecha) {
        this.idProgramaAcademico = id;
        this.nombre = nombre;
        this.descripcion = descripcion;
        this.fecha = fecha;
    }
```

Figure 4: ProgramaAcademico.java

La clase ProgramaAcad.java se encarga de la interfaz gráfica del usuario, permitiendo una interacción intuitiva con la aplicación.

```java
public class ProgramaAcad extends javax.swing.JFrame {


    private ProgramaAcademicoApp app;

    public ProgramaAcad() {
        initComponents();
            app = new ProgramaAcademicoApp();
            cargarDatosEnTabla();
            Guardar.setEnabled(false);
            Editar.setEnabled(false);
            Buscar.setEnabled(false);
            Cancelar.setEnabled(false);
            Clave.setEnabled(false);
            Nombre.setEnabled(false);
            Descripcion.setEnabled(false);
            Fecha.setEnabled(false);
            setTitle("Programas Académicos");

    }
```

Figure 5: Enter Caption

2.0.4 Funcionamiento de la Aplicación

1. Pantalla Principal

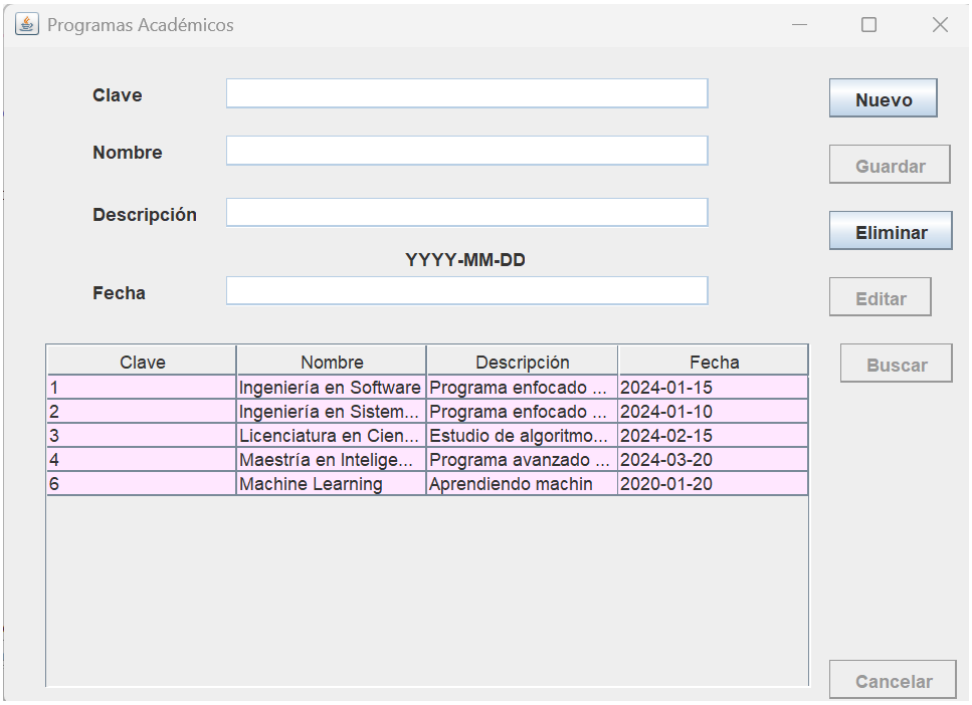

Figure 6: Pantalla Inicial

2. Agregar Programa Académico

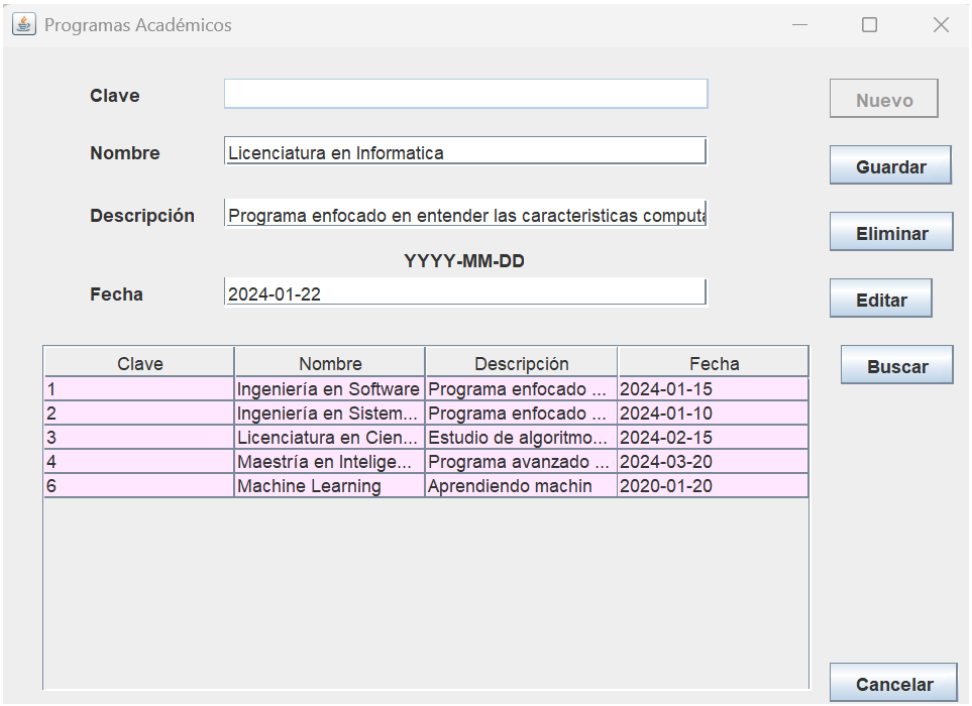

Figure 7: Botón Agregar

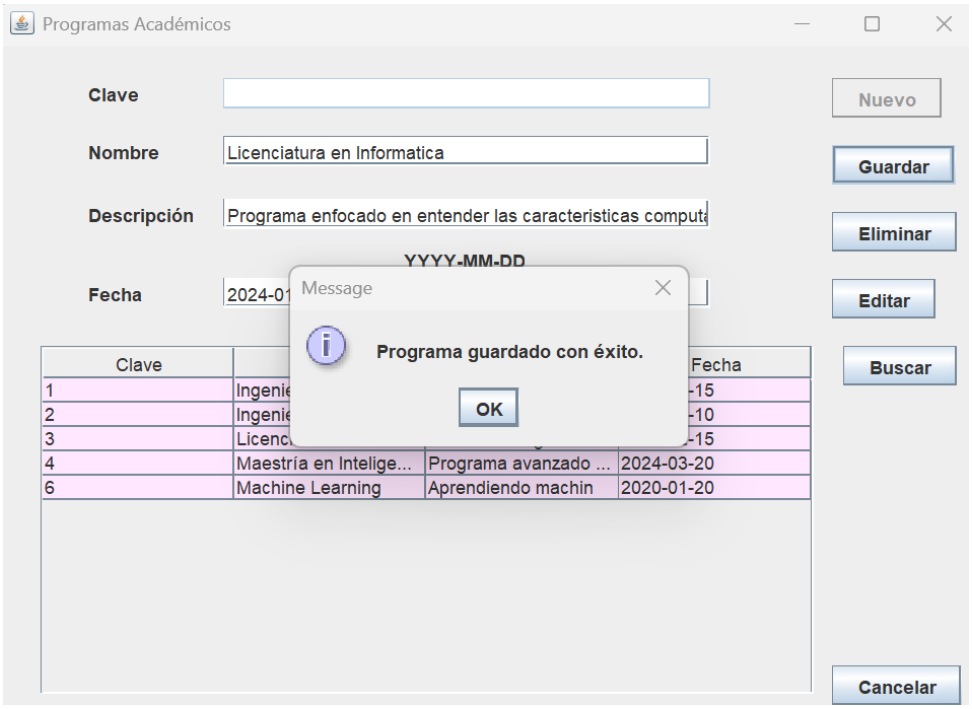

Figure 8: Agregado con Éxito

3. Editar Programa Académico

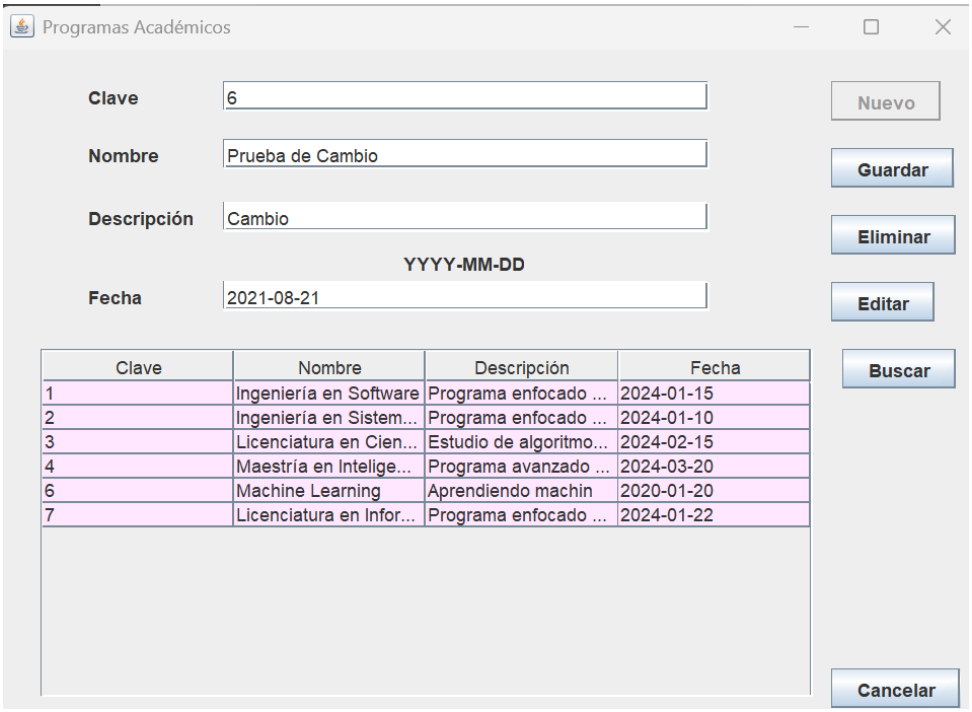

Figure 9: Botón Editar

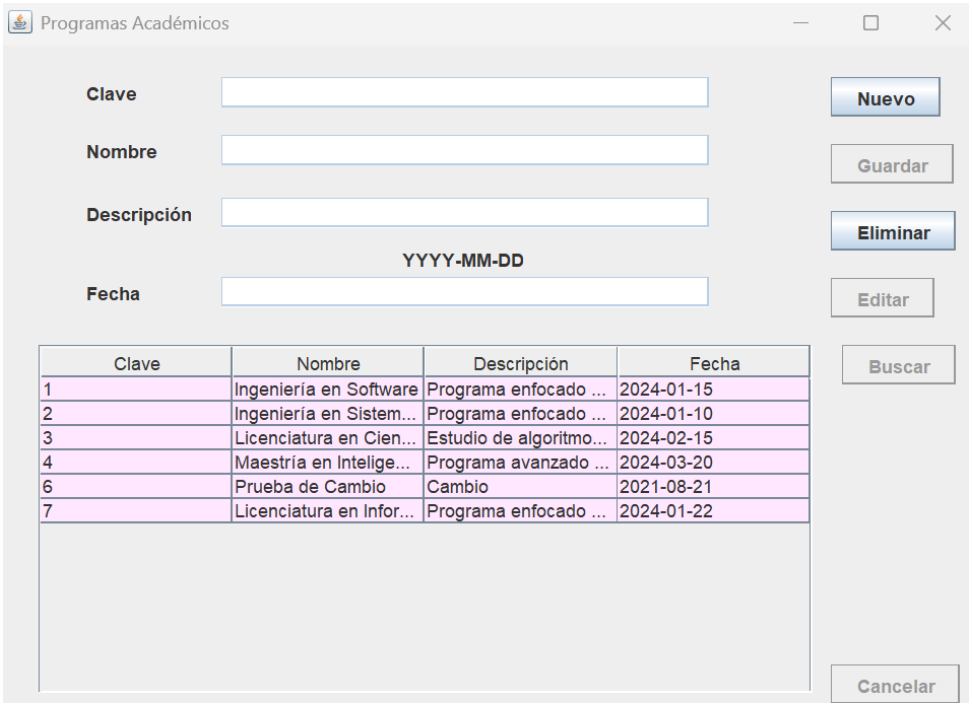

Figure 10: Resultado de la Edición

4. Eliminar Programa Académico

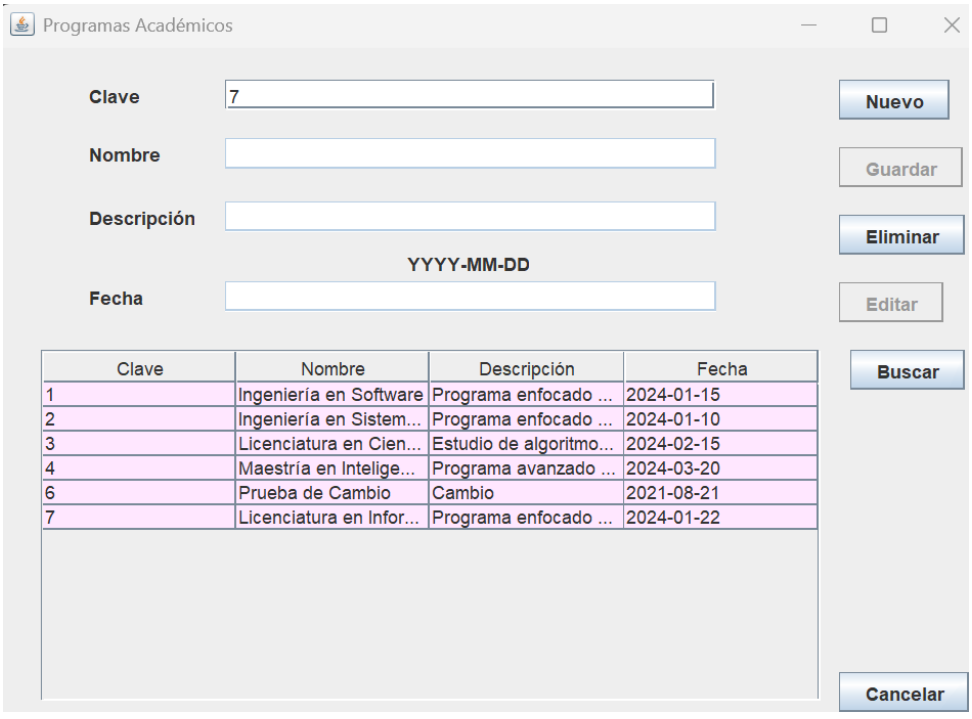

Figure 11: Botón Eliminar

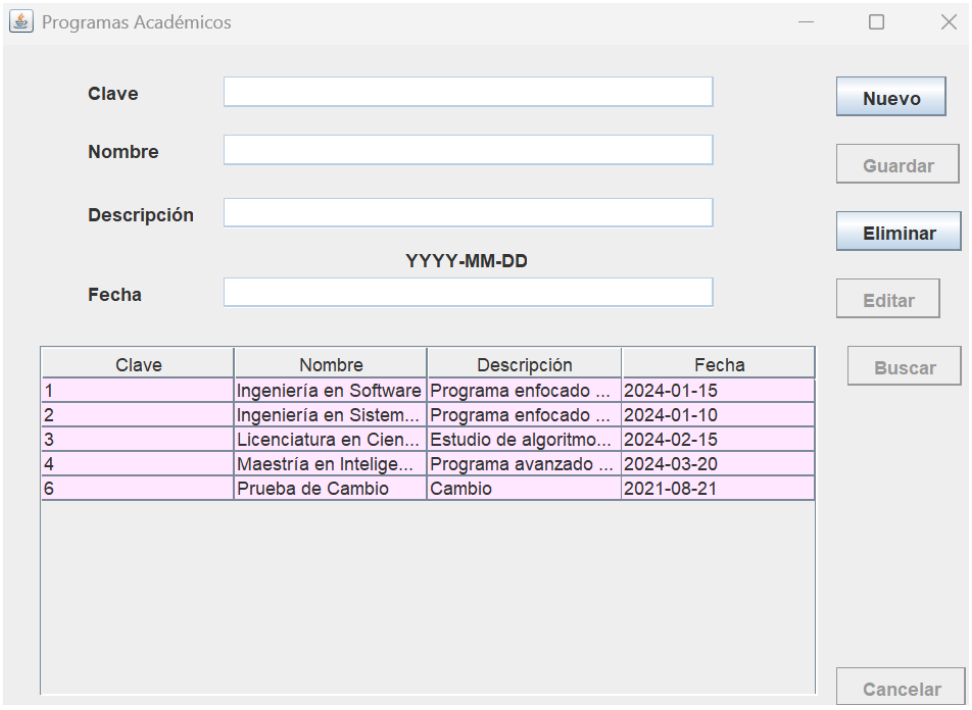

Figure 12: Resultado de Eliminar

5. Buscar Programa Académico

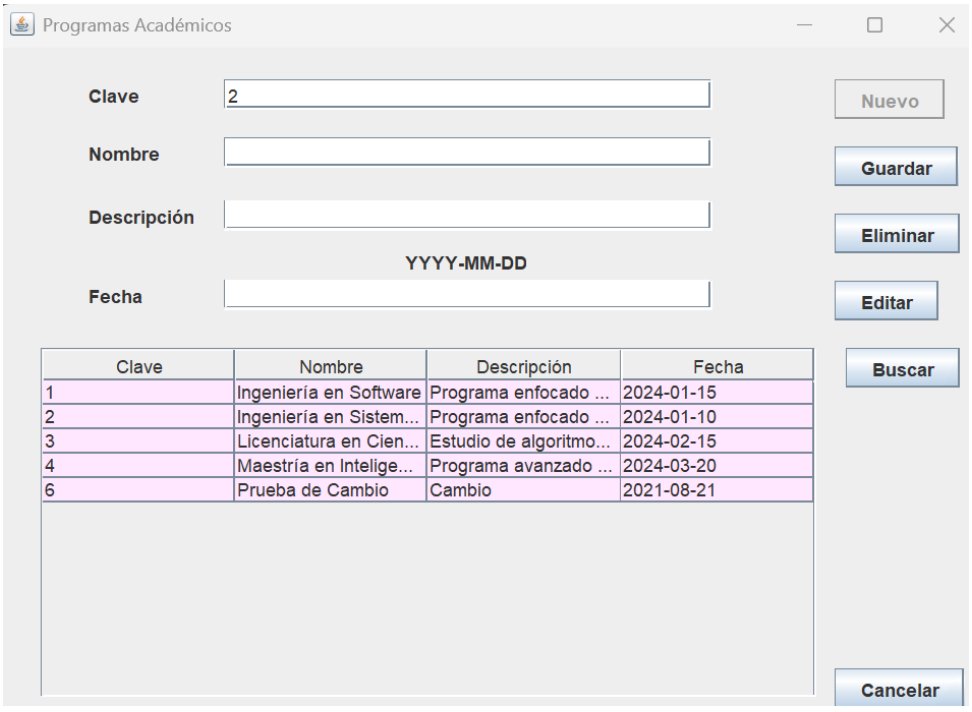

Figure 13: Botón Buscar

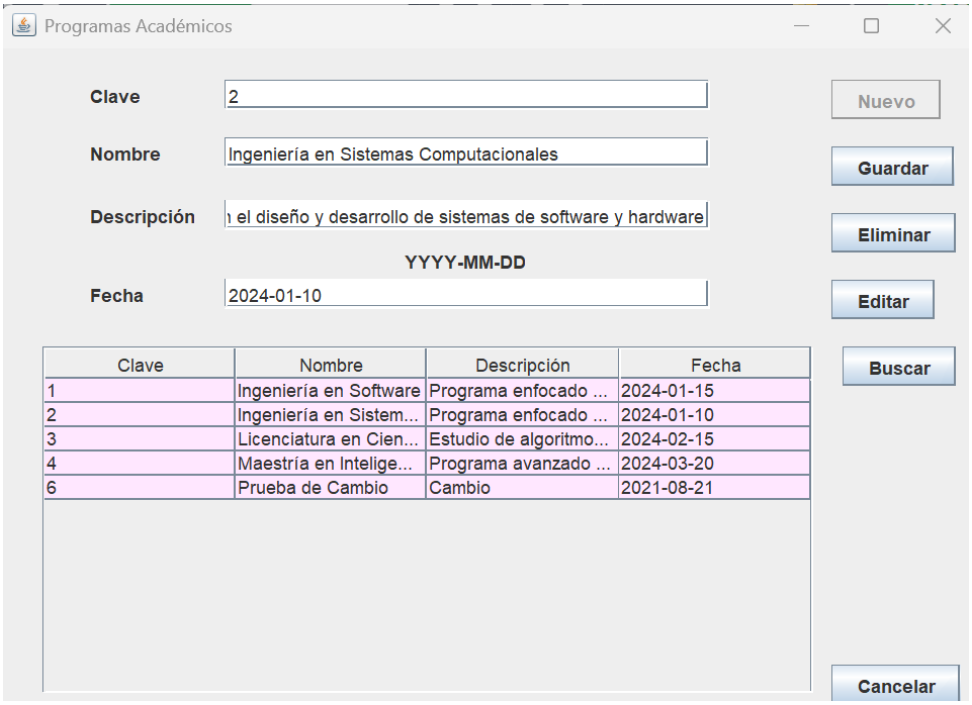

Figure 14: Resultado de Buscar

6. Cancelar Programa Académico

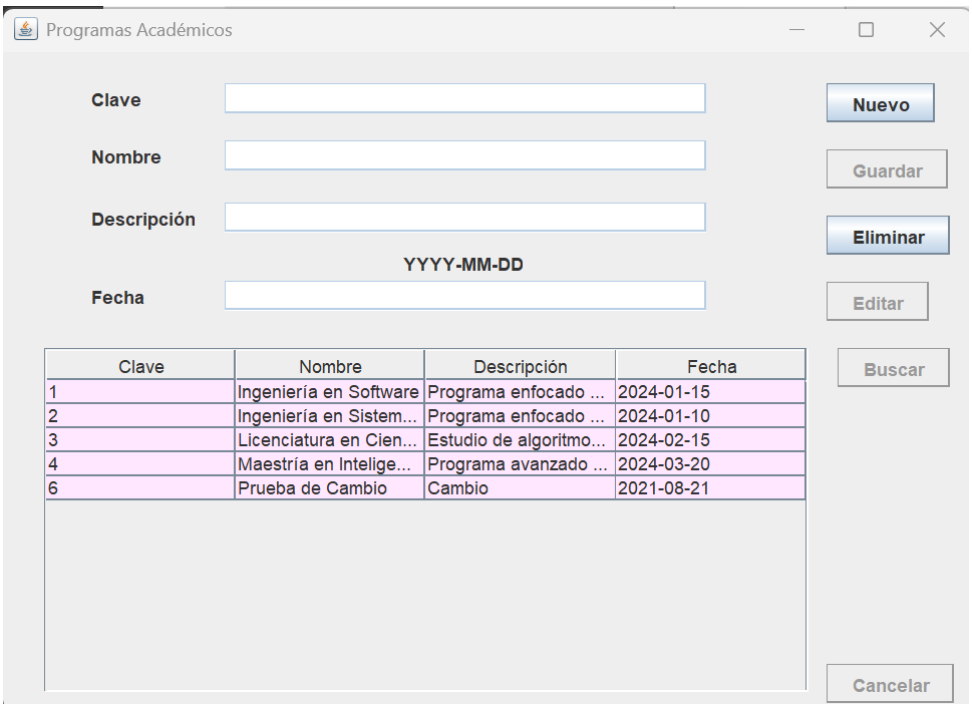

Figure 15: Botón Cancelar

3 Conclusión

El desarrollo de una aplicación que interactúa con una base de datos utilizando JDBC es una tarea fundamental en el ámbito del desarrollo de software. A lo largo de esta práctica, se han abordado conceptos clave como la conexión a bases de datos, la ejecución de consultas SQL y la gestión de transacciones, los cuales son esenciales para construir aplicaciones robustas y escalables.

Sin embargo, durante este proceso, es común enfrentarse a ciertos tipos de errores y desafíos que pueden afectar el funcionamiento de la aplicación. Algunos de los más comunes incluyen:

- **Errores de conexión:** Ocurren cuando la aplicación no puede establecer una conexión con la base de datos, debido a credenciales incorrectas, problemas de red o errores en la URL de conexión.
- **Errores de sintaxis SQL:** Se presentan cuando hay errores en las consultas, como palabras clave mal escritas o estructuras incorrectas.
- **Errores de tipo de datos:** Suceden cuando hay discrepancias entre los tipos de datos en la base de datos y los que maneja la aplicación.
- **Errores de transacciones:** Ocurren cuando no se gestionan adecuadamente las transacciones, lo que puede provocar inconsistencias en los datos.
- **Errores de recursos no cerrados:** Se presentan cuando no se cierran adecuadamente los objetos como `Connection`, `Statement` o `ResultSet`, lo que puede afectar el rendimiento.
- **Errores de concurrencia:** Ocurren cuando múltiples usuarios o procesos intentan acceder o modificar los mismos datos simultáneamente, generando posibles conflictos.

Para mitigar estos problemas, es importante aplicar buenas prácticas como:

- Verificar credenciales y configuraciones de conexión.
- Revisar las consultas SQL antes de ejecutarlas.
- Asegurar la compatibilidad entre los tipos de datos.
- Utilizar bloques `try-catch` y `rollback` en la gestión de transacciones.
- Cerrar adecuadamente los recursos para evitar fugas de memoria.
- Implementar mecanismos de control de concurrencia en la base de datos.

El manejo adecuado de estos errores no solo mejora la estabilidad y el rendimiento de la aplicación, sino que también contribuye a una experiencia de usuario más fluida y confiable. Comprender y anticipar estos errores permite desarrollar habilidades de depuración y resolución de problemas, esenciales en el desarrollo de software.

En conclusión, esta práctica no solo ha permitido implementar operaciones CRUD utilizando JDBC, sino que también ha brindado una visión clara de los desafíos comunes y las mejores prácticas para superarlos. Estos conocimientos son fundamentales para cualquier desarrollador que busque construir aplicaciones robustas y eficientes en Java.

Mendez Cruz Ivonne

4 Referencias Bibliográficas

References

[1] Gosling, J., Joy, B., Steele, G., Bracha, G., Buckley, A., "The Java Language Specification", Oracle America, Inc. Accedido el 11 de marzo de 2025. [En línea]. Disponible: https://docs.oracle.com/javase/specs/

[2] ISO/IEC 9075:2016, "Information technology – Database languages – SQL", International Organization for Standardization. Accedido el 11 de marzo de 2025. [En línea]. Disponible: https://www.iso.org/standard/63555.html