CARRERA DESARROLLO DE SOFTWARE

PROGRAMACIÓN ORIENTADA A OBJETOS

PROYECTO

CRISTINA SOLEDAD PROAÑO ULLAGUARI

cristina.proano@cenestur.edu.ec

QUITO 2025 -2026

Introducción

El presente ensayo tiene como objetivo explicar y analizar el desarrollo del Sistema de Gestión de Notas Académicas, un proyecto integral desarrollado en Java que implementa un sistema CRUD completo utilizando patrones de diseño orientados a objetos, interfaces gráficas con Swing, y persistencia de datos en MySQL. Este sistema representa la aplicación práctica de los conceptos fundamentales de la Programación Orientada a Objetos, abordando desde la arquitectura del software hasta la implementación específica de cada componente.

CODIGO PRINIPAL EJECUTADO

```
-- CREATE DATABASE IF NOT EXISTS sistema_notas;
-- USE sistema_notas;

-- TABLAS
-- Tabla Persona (información base)
CREATE TABLE IF NOT EXISTS Persona (
    id_persona INT AUTO_INCREMENT PRIMARY KEY,
    cedula VARCHAR(20) UNIQUE NOT NULL,
    nombres VARCHAR(50) NOT NULL,
    apellidos VARCHAR(50) NOT NULL,
    email VARCHAR(50) UNIQUE NOT NULL,
    telefono VARCHAR(15),
    tipo ENUM('Administrador', 'Profesor', 'Estudiante')
NOT NULL
);

-- Tabla Usuario (credenciales de acceso)
CREATE TABLE IF NOT EXISTS Usuario (
    id_usuario INT AUTO_INCREMENT PRIMARY KEY,
    id_persona INT UNIQUE NOT NULL,
    username VARCHAR(50) UNIQUE NOT NULL,
    password VARCHAR(100) NOT NULL,
    FOREIGN KEY (id_persona) REFERENCES
Persona(id_persona) ON DELETE CASCADE
);

-- Tabla Estudiante
CREATE TABLE IF NOT EXISTS Estudiante (
```

```
    id_estudiante INT PRIMARY KEY,
    matricula VARCHAR(20) UNIQUE NOT NULL,
    FOREIGN KEY (id_estudiante) REFERENCES
Persona(id_persona) ON DELETE CASCADE
);

-- Tabla Profesor
CREATE TABLE IF NOT EXISTS Profesor (
    id_profesor INT PRIMARY KEY,
    especialidad VARCHAR(50) NOT NULL,
    FOREIGN KEY (id_profesor) REFERENCES
Persona(id_persona) ON DELETE CASCADE
);

-- Tabla Materia
CREATE TABLE IF NOT EXISTS Materia (
    id_materia INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(50) NOT NULL,
    descripcion VARCHAR(200),
    creditos TINYINT NOT NULL DEFAULT 3
);

-- Tabla Asignación Profesor-Materia
CREATE TABLE IF NOT EXISTS ProfesorMateria (
    id_profesor INT NOT NULL,
    id_materia INT NOT NULL,
    periodo_academico VARCHAR(10) NOT NULL,
    PRIMARY KEY (id_profesor, id_materia,
periodo_academico),
    FOREIGN KEY (id_profesor) REFERENCES
Profesor(id_profesor) ON DELETE CASCADE,
    FOREIGN KEY (id_materia) REFERENCES
Materia(id_materia) ON DELETE CASCADE
);

-- Tabla EstudianteMateria
CREATE TABLE IF NOT EXISTS EstudianteMateria (
    id_estudiante INT NOT NULL,
    id_materia INT NOT NULL,
    periodo_academico VARCHAR(10) NOT NULL,
    PRIMARY KEY (id_estudiante, id_materia,
periodo_academico),
    FOREIGN KEY (id_estudiante) REFERENCES
Estudiante(id_estudiante) ON DELETE CASCADE,
    FOREIGN KEY (id_materia) REFERENCES
Materia(id_materia) ON DELETE CASCADE
);

-- Tabla Nota (versión simplificada)
CREATE TABLE IF NOT EXISTS Nota (
    id_nota INT AUTO_INCREMENT PRIMARY KEY,
```

```
    id_estudiante INT NOT NULL,
    id_materia INT NOT NULL,
    id_profesor INT NOT NULL,
    periodo ENUM('1Q', '2Q', '3Q', '4Q') NOT NULL,
    nota_parcial1 DECIMAL(5,2) CHECK (nota_parcial1
BETWEEN 0 AND 10),
    nota_parcial2 DECIMAL(5,2) CHECK (nota_parcial2
BETWEEN 0 AND 10),
    nota_final DECIMAL(5,2) CHECK (nota_final BETWEEN 0
AND 10),
    fecha_registro TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (id_estudiante) REFERENCES
Estudiante(id_estudiante) ON DELETE CASCADE,
    FOREIGN KEY (id_materia) REFERENCES
Materia(id_materia) ON DELETE CASCADE,
    FOREIGN KEY (id_profesor) REFERENCES
Profesor(id_profesor) ON DELETE CASCADE,
    UNIQUE KEY (id_estudiante, id_materia, periodo)
);

-- ============================================
-- DATOS INICIALES
-- ============================================

-- Personas
INSERT INTO Persona (cedula, nombres, apellidos, email,
telefono, tipo) VALUES
('0000000001', 'Admin', 'Principal', 'admin@escuela.com',
'0999999999', 'Administrador'),
('0000000002', 'Juan', 'Pérez', 'jperez@escuela.com',
'0988888888', 'Profesor'),
('1728394050', 'Carlos', 'López', 'clopez@escuela.com',
'0966666666', 'Estudiante');

-- Usuarios
INSERT INTO Usuario (id_persona, username, password)
VALUES
(1, 'admin', SHA2('admin123', 256)),
(2, 'jperez', SHA2('profe123', 256)),
(3, 'clopez', SHA2('estu123', 256));

-- Profesores
INSERT INTO Profesor (id_profesor, especialidad) VALUES
(2, 'Matemáticas');

-- Estudiantes
INSERT INTO Estudiante (id_estudiante, matricula) VALUES
(3, 'EST-2023-001');

-- Materias
```

```
INSERT INTO Materia (nombre, descripcion, creditos)
VALUES
('Matemáticas Básicas', 'Álgebra y geometría
fundamental', 4),
('Literatura Universal', 'Estudio de obras clásicas', 3);

-- Asignación Profesor-Materia
INSERT INTO ProfesorMateria (id_profesor, id_materia,
periodo_academico) VALUES
(2, 1, '2023-1Q');

-- Inscripción de estudiantes en materias
INSERT INTO EstudianteMateria (id_estudiante, id_materia,
periodo_academico) VALUES
(3, 1, '2023-1Q'),  -- Carlos López en Matemáticas
Básicas
(3, 2, '2023-1Q');  -- Carlos López en Literatura
Universal

-- Notas (CORREGIDO)
INSERT INTO Nota (id_estudiante, id_materia, id_profesor,
periodo, nota_parcial1, nota_parcial2, nota_final) VALUES
(3, 1, 2, '1Q', 8.5, 7.8, 8.2);
```

El proyecto implementa el patrón arquitectural MVC, que separa claramente las responsabilidades del sistema en tres capas bien definidas:

Modelo (Model): Contiene las clases que representan las entidades del dominio del problema. Aunque no se muestran completamente en los archivos proporcionados, podemos inferir la existencia de clases como Persona, Usuario, Estudiante, Profesor, Materia, y Nota. Estas clases encapsulan los datos y las reglas de negocio del sistema educativo.

Vista (View): Implementada mediante Java Swing, incluye interfaces como AuthView, AdminView, ProfesorView, y EstudianteView. Estas clases se encargan de la presentación de la información al usuario y la captura de sus interacciones.

Controlador (Controller): Actúa como intermediario entre el modelo y la vista. El AdminController analizado demuestra cómo se gestionan las acciones del usuario, se coordinan las operaciones con la base de datos a través de los DAOs, y se actualiza la interfaz gráfica según sea necesario.

Patrón Data Access Object (DAO)

El sistema implementa el patrón DAO para abstraer y encapsular todo el acceso a la fuente de datos. Cada entidad del sistema tiene su correspondiente clase DAO:

Conexión

```
package dao;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.Properties;

public class Conexion {
    //  CORREGIDO: URL completa con jdbc:mysql://,
puerto y base de datos
    private static final String DB_URL =
"jdbc:mysql://b3ud4ghs0ekyobcmgbsn-
mysql.services.clever-
cloud.com:3306/b3ud4ghs0ekyobcmgbsn";
    private static final String DB_USER =
"uok4pwobhxe8lmkg";
    private static final String DB_PASSWORD =
"GXtcyT0B81t61AcVtpfW";

    public static boolean testConnection() {
        Connection conn = null;
        try {
            //  AGREGADO: Cargar el driver de MySQL

Class.forName("com.mysql.cj.jdbc.Driver");

            System.out.println(" Intentando conectar
a: " + DB_URL);
            System.out.println(" Usuario: " +
DB_USER);

            // Configurar propiedades de conexión
```

```
            Properties props = new Properties();
            props.setProperty("user", DB_USER);
            props.setProperty("password",
DB_PASSWORD);
            props.setProperty("useSSL", "false");
            props.setProperty("serverTimezone",
"UTC");

props.setProperty("allowPublicKeyRetrieval", "true");

            // Intentar establecer conexión
            conn =
DriverManager.getConnection(DB_URL, props);

            if (conn != null && conn.isValid(2)) {
                System.out.println(" ¡Conexión
exitosa a Clever Cloud!");
                return true;
            }

        } catch (ClassNotFoundException e) {
            System.err.println("Error: Driver MySQL
no encontrado");
            e.printStackTrace();
            return false;
        } catch (SQLException e) {
            System.err.println("Error al probar la
conexión:");
            System.err.println("URL: " + DB_URL);
            System.err.println("Usuario: " +
DB_USER);
            System.err.println("Código de error: " +
e.getErrorCode());
            System.err.println("Mensaje de error: " +
e.getMessage());
            return false;
        } finally {
            // Cerrar conexión si existe
            if (conn != null) {
                try {
                    conn.close();
                } catch (SQLException e) {
                    System.err.println("Error al
cerrar la conexión de prueba: " + e.getMessage());
                }
            }
        }
        return false;
    }
```

```
    public static Connection getConnection() throws
SQLException {
        try {
            //  AGREGADO: Cargar el driver de MySQL

Class.forName("com.mysql.cj.jdbc.Driver");

            Properties props = new Properties();
            props.setProperty("user", DB_USER);
            props.setProperty("password",
DB_PASSWORD);
            props.setProperty("useSSL", "false");
            props.setProperty("serverTimezone",
"UTC");

props.setProperty("allowPublicKeyRetrieval", "true");

            return
DriverManager.getConnection(DB_URL, props);

        } catch (ClassNotFoundException e) {
            throw new SQLException("Driver MySQL no
encontrado", e);
        }
    }
}
```

UsuarioDAO: Gestiona las operaciones CRUD para usuarios del sistema

```
package dao;

import model.Usuario;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class UsuarioDAO {
    public List<Usuario> listarUsuarios() throws
SQLException {
        List<Usuario> usuarios = new ArrayList<>();
        String sql = "SELECT u.*, p.tipo as rol FROM
Usuario u JOIN Persona p ON u.id_persona = p.id_persona";

        try (Connection conn = Conexion.getConnection();
             Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery(sql)) {

            while (rs.next()) {
                usuarios.add(new Usuario(
```

```
                        rs.getInt("id_usuario"),
                        rs.getInt("id_persona"),
                        rs.getString("username"),
                        rs.getString("password"),
                        rs.getString("rol")));
            }
        }
        return usuarios;
    }

    public boolean crearUsuario(Usuario usuario) throws
SQLException {
        String sql = "INSERT INTO Usuario (id_persona,
username, password) VALUES (?, ?, SHA2(?, 256))";

        try (Connection conn = Conexion.getConnection();
             PreparedStatement stmt =
conn.prepareStatement(sql)) {

            stmt.setInt(1, usuario.getIdPersona());
            stmt.setString(2, usuario.getUsername());
            stmt.setString(3, usuario.getPassword());

            return stmt.executeUpdate() > 0;
        }
    }

    public Usuario buscarPorId(int idUsuario) throws
SQLException {
        String sql = "SELECT u.*, p.tipo as rol FROM
Usuario u JOIN Persona p ON u.id_persona = p.id_persona
WHERE u.id_usuario = ?";

        try (Connection conn = Conexion.getConnection();
             PreparedStatement stmt =
conn.prepareStatement(sql)) {

            stmt.setInt(1, idUsuario);

            try (ResultSet rs = stmt.executeQuery()) {
                if (rs.next()) {
                    return new Usuario(
                            rs.getInt("id_usuario"),
                            rs.getInt("id_persona"),
                            rs.getString("username"),
                            rs.getString("password"),
                            rs.getString("rol"));
                }
            }
        }
        return null;
```

```
    }

    public Usuario buscarPorIdPersona(int idPersona)
throws SQLException {
        String sql = "SELECT u.*, p.tipo as rol FROM
Usuario u JOIN Persona p ON u.id_persona = p.id_persona
WHERE u.id_persona = ?";

        try (Connection conn = Conexion.getConnection();
             PreparedStatement stmt =
conn.prepareStatement(sql)) {

            stmt.setInt(1, idPersona);

            try (ResultSet rs = stmt.executeQuery()) {
                if (rs.next()) {
                    return new Usuario(
                            rs.getInt("id_usuario"),
                            rs.getInt("id_persona"),
                            rs.getString("username"),
                            rs.getString("password"),
                            rs.getString("rol"));
                }
            }
        }
        return null;
    }

    public boolean actualizarUsuario(Usuario usuario)
throws SQLException {
        String sql = "UPDATE Usuario SET username = ?,
password = CASE WHEN ? = '' THEN password ELSE SHA2(?,
256) END WHERE id_usuario = ?";

        try (Connection conn = Conexion.getConnection();
             PreparedStatement stmt =
conn.prepareStatement(sql)) {

            stmt.setString(1, usuario.getUsername());
            stmt.setString(2, usuario.getPassword());
            stmt.setString(3, usuario.getPassword());
            stmt.setInt(4, usuario.getIdUsuario());

            return stmt.executeUpdate() > 0;
        }
    }

    public boolean eliminarUsuario(int idUsuario) throws
SQLException {
        String sql = "DELETE FROM Usuario WHERE
id_usuario = ?";
```

```
        try (Connection conn = Conexion.getConnection();
             PreparedStatement stmt =
conn.prepareStatement(sql)) {

            stmt.setInt(1, idUsuario);
            return stmt.executeUpdate() > 0;
        }
    }

    public boolean registrarUsuario(Usuario usuario)
throws SQLException {
        String sql = "INSERT INTO Usuario (id_persona,
username, password) VALUES (?, ?, SHA2(?, 256))";

        try (Connection conn = Conexion.getConnection();
             PreparedStatement stmt =
conn.prepareStatement(sql,
Statement.RETURN_GENERATED_KEYS)) {

            stmt.setInt(1, usuario.getIdPersona());
            stmt.setString(2, usuario.getUsername());
            stmt.setString(3, usuario.getPassword());

            int filasAfectadas = stmt.executeUpdate();

            if (filasAfectadas > 0) {
                try (ResultSet generatedKeys =
stmt.getGeneratedKeys()) {
                    if (generatedKeys.next()) {

usuario.setIdUsuario(generatedKeys.getInt(1));
                        return true;
                    }
                }
            }
            return false;
        }
    }

    public Usuario autenticarUsuario(String username,
String password) throws SQLException {
        String sql = "SELECT u.*, p.tipo as rol FROM
Usuario u " +
                "JOIN Persona p ON u.id_persona =
p.id_persona " +
                "WHERE u.username = ? AND u.password =
SHA2(?, 256)";

        try (Connection conn = Conexion.getConnection();
```

```
             PreparedStatement stmt =
conn.prepareStatement(sql)) {

            stmt.setString(1, username);
            stmt.setString(2, password);

            try (ResultSet rs = stmt.executeQuery()) {
                if (rs.next()) {
                    return new Usuario(
                            rs.getInt("id_usuario"),
                            rs.getInt("id_persona"),
                            rs.getString("username"),
                            rs.getString("password"),
                            rs.getString("rol")
                    );
                }
            }
        }
        return null;
    }

    public List<Usuario> listarTodos() throws
SQLException {
        List<Usuario> usuarios = new ArrayList<>();
        String sql = "SELECT u.*, p.tipo as rol FROM
Usuario u JOIN Persona p ON u.id_persona = p.id_persona";

        try (Connection conn = Conexion.getConnection();
             Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery(sql)) {

            while (rs.next()) {
                usuarios.add(new Usuario(
                        rs.getInt("id_usuario"),
                        rs.getInt("id_persona"),
                        rs.getString("username"),
                        rs.getString("password"),
                        rs.getString("rol")
                ));
            }
        }
        return usuarios;
    }
}
```

PersonaDAO: Maneja la información personal base

```
package dao;
```

```
import model.Persona;
import java.sql.*;

public class PersonaDAO {
    public int registrarPersona(Persona persona) throws
SQLException {
        String sql = "INSERT INTO Persona (cedula,
nombres, apellidos, email, telefono, tipo) VALUES (?, ?,
?, ?, ?, ?)";

        try (Connection conn = Conexion.getConnection();
             PreparedStatement stmt =
conn.prepareStatement(sql,
Statement.RETURN_GENERATED_KEYS)) {

            stmt.setString(1, persona.getCedula());
            stmt.setString(2, persona.getNombres());
            stmt.setString(3, persona.getApellidos());
            stmt.setString(4, persona.getEmail());
            stmt.setString(5, persona.getTelefono());
            stmt.setString(6, persona.getTipo());

            int affectedRows = stmt.executeUpdate();

            if (affectedRows > 0) {
                try (ResultSet rs =
stmt.getGeneratedKeys()) {
                    if (rs.next()) {
                        return rs.getInt(1);
                    }
                }
            }
            return 0;
        }
    }

    public Persona obtenerPorId(int idPersona) throws
SQLException {
        String sql = "SELECT * FROM Persona WHERE
id_persona = ?";

        try (Connection conn = Conexion.getConnection();
             PreparedStatement stmt =
conn.prepareStatement(sql)) {

            stmt.setInt(1, idPersona);

            try (ResultSet rs = stmt.executeQuery()) {
                if (rs.next()) {
                    return new Persona(
                            rs.getInt("id_persona"),
```

```
                                rs.getString("cedula"),
                                rs.getString("nombres"),
                                rs.getString("apellidos"),
                                rs.getString("email"),
                                rs.getString("telefono"),
                                rs.getString("tipo")
                        );
                    }
                }
            }
            return null;
        }

        public boolean eliminarPersona(int idPersona) throws
SQLException {
            String sql = "DELETE FROM Persona WHERE
id_persona = ?";

            try (Connection conn = Conexion.getConnection();
                 PreparedStatement stmt =
conn.prepareStatement(sql)) {

                stmt.setInt(1, idPersona);
                return stmt.executeUpdate() > 0;
            }
        }

        public void actualizarPersona(Persona persona) {
        }
}
```

ProfesorDAO: Operaciones específicas para profesores

```
import model.Profesor;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class ProfesorDAO {
    public boolean registrarProfesor(Profesor profesor)
throws SQLException {
        String sql = "INSERT INTO Profesor (id_profesor,
especialidad) VALUES (?, ?)";

        try (Connection conn = Conexion.getConnection();
             PreparedStatement stmt =
conn.prepareStatement(sql)) {

            stmt.setInt(1, profesor.getIdProfesor());
```

```
            stmt.setString(2,
profesor.getEspecialidad());

            return stmt.executeUpdate() > 0;
        }
    }

    public boolean eliminarProfesor(int idProfesor)
throws SQLException {
        String sql = "DELETE FROM Profesor WHERE
id_profesor = ?";

        try (Connection conn = Conexion.getConnection();
             PreparedStatement stmt =
conn.prepareStatement(sql)) {

            stmt.setInt(1, idProfesor);
            return stmt.executeUpdate() > 0;
        }
    }

    public List<Profesor> listarTodos() throws
SQLException {
        String sql = "SELECT p.id_profesor,
p.especialidad, per.nombres, per.apellidos " +
                "FROM Profesor p " +
                "JOIN Persona per ON p.id_profesor =
per.id_persona";

        List<Profesor> profesores = new ArrayList<>();

        try (Connection conn = Conexion.getConnection();
             PreparedStatement stmt =
conn.prepareStatement(sql);
             ResultSet rs = stmt.executeQuery()) {

            while (rs.next()) {
                Profesor profesor = new Profesor(
                        rs.getInt("id_profesor"),
                        rs.getString("especialidad")
                );

profesor.setNombres(rs.getString("nombres"));

profesor.setApellidos(rs.getString("apellidos"));
                profesores.add(profesor);
            }
        }
        return profesores;
    }
```

```
    public Profesor obtenerPorId(int idProfesor) throws
SQLException {
        String sql = "SELECT p.especialidad, per.nombres,
per.apellidos " +
                "FROM Profesor p " +
                "JOIN Persona per ON p.id_profesor =
per.id_persona " +
                "WHERE p.id_profesor = ?";

        try (Connection conn = Conexion.getConnection();
             PreparedStatement stmt =
conn.prepareStatement(sql)) {

            stmt.setInt(1, idProfesor);
            ResultSet rs = stmt.executeQuery();

            if (rs.next()) {
                Profesor profesor = new Profesor(
                        idProfesor,
                        rs.getString("especialidad")
                );

profesor.setNombres(rs.getString("nombres"));

profesor.setApellidos(rs.getString("apellidos"));
                return profesor;
            }
        }
        return null;
    }

    public Profesor obtenerPorIdPersona(int idPersona)
throws SQLException {
        String sql = "SELECT p.id_profesor,
p.especialidad, per.nombres, per.apellidos " +
                "FROM Profesor p " +
                "JOIN Persona per ON p.id_profesor =
per.id_persona " +
                "WHERE p.id_profesor = ?";

        try (Connection conn = Conexion.getConnection();
             PreparedStatement stmt =
conn.prepareStatement(sql)) {

            stmt.setInt(1, idPersona);
            ResultSet rs = stmt.executeQuery();

            if (rs.next()) {
                Profesor profesor = new Profesor(
                        rs.getInt("id_profesor"),
                        rs.getString("especialidad")
```

```
                );
profesor.setNombres(rs.getString("nombres"));

profesor.setApellidos(rs.getString("apellidos"));
                return profesor;
            }
        }
        return null;
    }
```

EstudianteDAO: Gestiona estudiantes y sus notas

```
package dao;

import model.Estudiante;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class EstudianteDAO {
    public boolean registrarEstudiante(Estudiante
estudiante) throws SQLException {
        String sql = "INSERT INTO Estudiante
(id_estudiante, matricula) VALUES (?, ?)";

        try (Connection conn = Conexion.getConnection();
             PreparedStatement stmt =
conn.prepareStatement(sql)) {

            stmt.setInt(1, estudiante.getIdEstudiante());
            stmt.setString(2, estudiante.getMatricula());

            return stmt.executeUpdate() > 0;
        }
    }

    public boolean eliminarEstudiante(int idEstudiante)
throws SQLException {
        String sql = "DELETE FROM Estudiante WHERE
id_estudiante = ?";

        try (Connection conn = Conexion.getConnection();
             PreparedStatement stmt =
conn.prepareStatement(sql)) {

            stmt.setInt(1, idEstudiante);
            return stmt.executeUpdate() > 0;
        }
    }
```

```
    public List<Estudiante> listarTodos() throws
SQLException {
        String sql = "SELECT e.id_estudiante,
e.matricula, p.nombres, p.apellidos, p.email " +
                "FROM Estudiante e JOIN Persona p ON
e.id_estudiante = p.id_persona";

        List<Estudiante> estudiantes = new ArrayList<>();

        try (Connection conn = Conexion.getConnection();
             PreparedStatement stmt =
conn.prepareStatement(sql);
             ResultSet rs = stmt.executeQuery()) {

            while (rs.next()) {
                estudiantes.add(new Estudiante(
                        rs.getInt("id_estudiante"),
                        rs.getString("matricula"),
                        rs.getString("nombres"),
                        rs.getString("apellidos"),
                        rs.getString("email")
                ));
            }
        }
        return estudiantes;
    }

    public List<Estudiante>
listarEstudiantesPorMateria(int idMateria) throws
SQLException {
        String sql = "SELECT e.id_estudiante,
e.matricula, p.nombres, p.apellidos, p.email, " +
                "n.nota_parcial1, n.nota_parcial2,
n.nota_final " +
                "FROM Estudiante e " +
                "JOIN Persona p ON e.id_estudiante =
p.id_persona " +
                "JOIN EstudianteMateria em ON
e.id_estudiante = em.id_estudiante " +
                "LEFT JOIN Nota n ON e.id_estudiante =
n.id_estudiante AND em.id_materia = n.id_materia " +
                "WHERE em.id_materia = ?";

        List<Estudiante> estudiantes = new ArrayList<>();

        try (Connection conn = Conexion.getConnection();
             PreparedStatement stmt =
conn.prepareStatement(sql)) {

            stmt.setInt(1, idMateria);
            ResultSet rs = stmt.executeQuery();
```

```
            while (rs.next()) {
                Estudiante estudiante = new Estudiante(
                        rs.getInt("id_estudiante"),
                        rs.getString("matricula"),
                        rs.getString("nombres"),
                        rs.getString("apellidos"),
                        rs.getString("email")
                );

                // Asignar notas si existen

estudiante.setNotaParcial1(rs.getDouble("nota_parcial1"))
;
                if (rs.wasNull())
estudiante.setNotaParcial1(null);


estudiante.setNotaParcial2(rs.getDouble("nota_parcial2"))
;
                if (rs.wasNull())
estudiante.setNotaParcial2(null);


estudiante.setNotaFinal(rs.getDouble("nota_final"));
                if (rs.wasNull())
estudiante.setNotaFinal(null);

                estudiantes.add(estudiante);
            }
        }
        return estudiantes;
    }

    public Estudiante buscarPorId(int idEstudiante)
throws SQLException {
        String sql = "SELECT e.id_estudiante,
e.matricula, p.nombres, p.apellidos, p.email " +
                "FROM Estudiante e JOIN Persona p ON
e.id_estudiante = p.id_persona " +
                "WHERE e.id_estudiante = ?";

        try (Connection conn = Conexion.getConnection();
             PreparedStatement stmt =
conn.prepareStatement(sql)) {

            stmt.setInt(1, idEstudiante);
            ResultSet rs = stmt.executeQuery();

            if (rs.next()) {
                return new Estudiante(
```

```
                        rs.getInt("id_estudiante"),
                        rs.getString("matricula"),
                        rs.getString("nombres"),
                        rs.getString("apellidos"),
                        rs.getString("email")
                );
            }
            return null;
        }
    }

    public boolean actualizarNotas(int idEstudiante, int
idMateria, int idProfesor,
                                   String periodo, Double
notaParcial1,
                                   Double notaParcial2,
Double notaFinal) throws SQLException {
        // Verificar si ya existe un registro de notas
para este estudiante en esta materia y periodo
        String sqlCheck = "SELECT 1 FROM Nota WHERE
id_estudiante = ? AND id_materia = ? AND periodo = ?";
        String sqlInsert = "INSERT INTO Nota
(id_estudiante, id_materia, id_profesor, periodo, " +
                "nota_parcial1, nota_parcial2,
nota_final) VALUES (?, ?, ?, ?, ?, ?, ?)";
        String sqlUpdate = "UPDATE Nota SET nota_parcial1
= ?, nota_parcial2 = ?, nota_final = ?, " +
                "id_profesor = ? WHERE id_estudiante = ?
AND id_materia = ? AND periodo = ?";

        try (Connection conn = Conexion.getConnection())
{
            // Verificar si existe registro previo
            boolean existeRegistro = false;
            try (PreparedStatement stmt =
conn.prepareStatement(sqlCheck)) {
                stmt.setInt(1, idEstudiante);
                stmt.setInt(2, idMateria);
                stmt.setString(3, periodo);
                try (ResultSet rs = stmt.executeQuery())
{
                    existeRegistro = rs.next();
                }
            }

            // Insertar o actualizar según corresponda
            if (existeRegistro) {
                try (PreparedStatement stmt =
conn.prepareStatement(sqlUpdate)) {
                    stmt.setDouble(1, notaParcial1);
                    stmt.setDouble(2, notaParcial2);
```

```
                    stmt.setDouble(3, notaFinal);
                    stmt.setInt(4, idProfesor);
                    stmt.setInt(5, idEstudiante);
                    stmt.setInt(6, idMateria);
                    stmt.setString(7, periodo);
                    return stmt.executeUpdate() > 0;
                }
            } else {
                try (PreparedStatement stmt =
conn.prepareStatement(sqlInsert)) {
                    stmt.setInt(1, idEstudiante);
                    stmt.setInt(2, idMateria);
                    stmt.setInt(3, idProfesor);
                    stmt.setString(4, periodo);
                    stmt.setDouble(5, notaParcial1);
                    stmt.setDouble(6, notaParcial2);
                    stmt.setDouble(7, notaFinal);
                    return stmt.executeUpdate() > 0;
                }
            }
        }
    }

    public boolean matricularEstudiante(int idEstudiante,
int idMateria, String periodo) throws SQLException {
        String sql = "INSERT INTO EstudianteMateria
(id_estudiante, id_materia, periodo_academico) VALUES (?,
?, ?)";

        try (Connection conn = Conexion.getConnection();
             PreparedStatement stmt =
conn.prepareStatement(sql)) {

            stmt.setInt(1, idEstudiante);
            stmt.setInt(2, idMateria);
            stmt.setString(3, periodo);

            return stmt.executeUpdate() > 0;
        }
    }

    public boolean desmatricularEstudiante(int
idEstudiante, int idMateria, String periodo) throws
SQLException {
        String sql = "DELETE FROM EstudianteMateria WHERE
id_estudiante = ? AND id_materia = ? AND
periodo_academico = ?";

        try (Connection conn = Conexion.getConnection();
             PreparedStatement stmt =
conn.prepareStatement(sql)) {
```

```
            stmt.setInt(1, idEstudiante);
            stmt.setInt(2, idMateria);
            stmt.setString(3, periodo);

            return stmt.executeUpdate() > 0;
        }
    }
    public List<Estudiante> obtenerMateriasConNotas(int
idEstudiante) throws SQLException {
        String sql = "SELECT m.id_materia, m.nombre,
m.creditos, " +
                "n.nota_parcial1, n.nota_parcial2,
n.nota_final " +
                "FROM Materia m " +
                "JOIN EstudianteMateria em ON
m.id_materia = em.id_materia " +
                "LEFT JOIN Nota n ON m.id_materia =
n.id_materia AND em.id_estudiante = n.id_estudiante " +
                "WHERE em.id_estudiante = ?";

        List<Estudiante> materias = new ArrayList<>();

        try (Connection conn = Conexion.getConnection();
             PreparedStatement stmt =
conn.prepareStatement(sql)) {

            stmt.setInt(1, idEstudiante);
            ResultSet rs = stmt.executeQuery();

            while (rs.next()) {
                Estudiante estudiante = new Estudiante(
                        idEstudiante,
                        "", // Matrícula no necesaria
aquí
                        "", // Nombres no necesarios
                        "", // Apellidos no necesarios
                        rs.getDouble("nota_parcial1"),
                        rs.getDouble("nota_parcial2"),
                        rs.getDouble("nota_final")
                );

                // Configurar notas (manejar NULL)

estudiante.setNotaParcial1(rs.getObject("nota_parcial1")
!= null ? rs.getDouble("nota_parcial1") : null);

estudiante.setNotaParcial2(rs.getObject("nota_parcial2")
!= null ? rs.getDouble("nota_parcial2") : null);
```

```
estudiante.setNotaFinal(rs.getObject("nota_final") !=
null ? rs.getDouble("nota_final") : null);

                materias.add(estudiante);
            }
        }
        return materias;
    }
    public Estudiante obtenerNotasEstudianteMateria(int
idEstudiante, int idMateria) throws SQLException {
        String sql = "SELECT nota_parcial1,
nota_parcial2, nota_final " +
                "FROM Nota " +
                "WHERE id_estudiante = ? AND id_materia =
?";

        Estudiante estudiante = new
Estudiante(idEstudiante, "");

        try (Connection conn = Conexion.getConnection();
             PreparedStatement stmt =
conn.prepareStatement(sql)) {

            stmt.setInt(1, idEstudiante);
            stmt.setInt(2, idMateria);
            ResultSet rs = stmt.executeQuery();

            if (rs.next()) {

estudiante.setNotaParcial1(rs.getObject("nota_parcial1")
!= null ? rs.getDouble("nota_parcial1") : null);

estudiante.setNotaParcial2(rs.getObject("nota_parcial2")
!= null ? rs.getDouble("nota_parcial2") : null);

estudiante.setNotaFinal(rs.getObject("nota_final") !=
null ? rs.getDouble("nota_final") : null);
            }
        }

        return estudiante;
    }
```

MateriaDAO: Administra las materias del sistema

```
package dao;

import model.Materia;
```

```
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class MateriaDAO {
    public List<Materia> listarMateriasPorProfesor(int
idProfesor) throws SQLException {
        List<Materia> materias = new ArrayList<>();
        String sql = "SELECT m.id_materia, m.nombre,
m.descripcion, m.creditos " +
                "FROM Materia m JOIN ProfesorMateria pm
ON m.id_materia = pm.id_materia " +
                "WHERE pm.id_profesor = ?";

        try (Connection conn = Conexion.getConnection();
             PreparedStatement stmt =
conn.prepareStatement(sql)) {

            stmt.setInt(1, idProfesor);

            try (ResultSet rs = stmt.executeQuery()) {
                while (rs.next()) {
                    materias.add(new Materia(
                            rs.getInt("id_materia"),
                            rs.getString("nombre"),
                            rs.getString("descripcion"),
                            rs.getInt("creditos")
                    ));
                }
            }
        }
        return materias;
    }

    public List<String>
listarNombresMateriasPorProfesor(int idProfesor) throws
SQLException {
        List<String> nombresMaterias = new ArrayList<>();
        String sql = "SELECT m.nombre FROM Materia m " +
                "JOIN ProfesorMateria pm ON m.id_materia
= pm.id_materia " +
                "WHERE pm.id_profesor = ? " +
                "ORDER BY m.nombre";

        try (Connection conn = Conexion.getConnection();
             PreparedStatement stmt =
conn.prepareStatement(sql)) {

            stmt.setInt(1, idProfesor);

            try (ResultSet rs = stmt.executeQuery()) {
```

```
                while (rs.next()) {

nombresMaterias.add(rs.getString("nombre"));
                }
            }
        }
        return nombresMaterias;
    }

    public int obtenerIdMateriaPorNombre(String
nombreMateria) throws SQLException {
        String sql = "SELECT id_materia FROM Materia
WHERE nombre = ?";

        try (Connection conn = Conexion.getConnection();
             PreparedStatement stmt =
conn.prepareStatement(sql)) {

            stmt.setString(1, nombreMateria);

            try (ResultSet rs = stmt.executeQuery()) {
                if (rs.next()) {
                    return rs.getInt("id_materia");
                }
            }
        }
        return -1;
    }

    public Materia obtenerPorNombre(String nombreMateria)
throws SQLException {
        String sql = "SELECT id_materia, nombre,
descripcion, creditos FROM Materia WHERE nombre = ?";

        try (Connection conn = Conexion.getConnection();
             PreparedStatement stmt =
conn.prepareStatement(sql)) {

            stmt.setString(1, nombreMateria);

            try (ResultSet rs = stmt.executeQuery()) {
                if (rs.next()) {
                    return new Materia(
                            rs.getInt("id_materia"),
                            rs.getString("nombre"),
                            rs.getString("descripcion"),
                            rs.getInt("creditos")
                    );
                }
            }
        }
```

```
        return null;
    }

    public List<Materia> listarTodas() throws
SQLException {
        List<Materia> materias = new ArrayList<>();
        String sql = "SELECT id_materia, nombre,
descripcion, creditos FROM Materia";

        try (Connection conn = Conexion.getConnection();
             Statement stmt = conn.createStatement();
             ResultSet rs = stmt.executeQuery(sql)) {

            while (rs.next()) {
                materias.add(new Materia(
                        rs.getInt("id_materia"),
                        rs.getString("nombre"),
                        rs.getString("descripcion"),
                        rs.getInt("creditos")
                ));
            }
        }
        return materias;
    }

    public Materia obtenerPorId(int idMateria) throws
SQLException {
        String sql = "SELECT nombre, descripcion,
creditos FROM Materia WHERE id_materia = ?";

        try (Connection conn = Conexion.getConnection();
             PreparedStatement stmt =
conn.prepareStatement(sql)) {

            stmt.setInt(1, idMateria);

            try (ResultSet rs = stmt.executeQuery()) {
                if (rs.next()) {
                    return new Materia(
                            idMateria,
                            rs.getString("nombre"),
                            rs.getString("descripcion"),
                            rs.getInt("creditos")
                    );
                }
            }
        }
        return null;
    }
}
```

ProfesorMateriaDAO: Maneja las asignaciones profesor-materia

```
package dao;

import model.ProfesorMateria;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class ProfesorMateriaDAO {
    public boolean asignarMateriaAProfesor(int
idProfesor, int idMateria, String periodo) throws
SQLException {
        String sql = "INSERT INTO ProfesorMateria
(id_profesor, id_materia, periodo_academico) VALUES (?,
?, ?)";

        try (Connection conn = Conexion.getConnection();
             PreparedStatement stmt =
conn.prepareStatement(sql)) {

            stmt.setInt(1, idProfesor);
            stmt.setInt(2, idMateria);
            stmt.setString(3, periodo);

            return stmt.executeUpdate() > 0;
        }
    }

    public boolean estaProfesorAsignado(int idProfesor,
int idMateria) throws SQLException {
        String sql = "SELECT 1 FROM ProfesorMateria WHERE
id_profesor = ? AND id_materia = ?";

        try (Connection conn = Conexion.getConnection();
             PreparedStatement stmt =
conn.prepareStatement(sql)) {

            stmt.setInt(1, idProfesor);
            stmt.setInt(2, idMateria);

            try (ResultSet rs = stmt.executeQuery()) {
                return rs.next();
            }
        }
    }

    public boolean desasignarMateriaDeProfesor(int
idProfesor, int idMateria, String periodo) throws
SQLException {
```

```
        String sql = "DELETE FROM ProfesorMateria WHERE
id_profesor = ? AND id_materia = ? AND periodo_academico
= ?";

        try (Connection conn = Conexion.getConnection();
             PreparedStatement stmt =
conn.prepareStatement(sql)) {

            stmt.setInt(1, idProfesor);
            stmt.setInt(2, idMateria);
            stmt.setString(3, periodo);

            return stmt.executeUpdate() > 0;
        }
    }

    public List<ProfesorMateria> listarTodas() throws
SQLException {
        String sql = "SELECT id_profesor, id_materia,
periodo_academico FROM ProfesorMateria";
        List<ProfesorMateria> asignaciones = new
ArrayList<>();

        try (Connection conn = Conexion.getConnection();
             PreparedStatement stmt =
conn.prepareStatement(sql);
             ResultSet rs = stmt.executeQuery()) {

            while (rs.next()) {
                ProfesorMateria pm = new ProfesorMateria(
                        rs.getInt("id_profesor"),
                        rs.getInt("id_materia"),
                        rs.getString("periodo_academico")
                );
                asignaciones.add(pm);
            }
        }
        return asignaciones;
    }

    public List<ProfesorMateria> listarPorProfesor(int
idProfesor) throws SQLException {
        String sql = "SELECT id_materia,
periodo_academico FROM ProfesorMateria WHERE id_profesor
= ?";
        List<ProfesorMateria> asignaciones = new
ArrayList<>();

        try (Connection conn = Conexion.getConnection();
             PreparedStatement stmt =
conn.prepareStatement(sql)) {
```

```
            stmt.setInt(1, idProfesor);
            ResultSet rs = stmt.executeQuery();

            while (rs.next()) {
                ProfesorMateria pm = new ProfesorMateria(
                        idProfesor,
                        rs.getInt("id_materia"),
                        rs.getString("periodo_academico")
                );
                asignaciones.add(pm);
            }
        }
        return asignaciones;
    }

    public List<ProfesorMateria> listarPorMateria(int
idMateria) throws SQLException {
        String sql = "SELECT id_profesor,
periodo_academico FROM ProfesorMateria WHERE id_materia =
?";
        List<ProfesorMateria> asignaciones = new
ArrayList<>();

        try (Connection conn = Conexion.getConnection();
             PreparedStatement stmt =
conn.prepareStatement(sql)) {

            stmt.setInt(1, idMateria);
            ResultSet rs = stmt.executeQuery();

            while (rs.next()) {
                ProfesorMateria pm = new ProfesorMateria(
                        rs.getInt("id_profesor"),
                        idMateria,
                        rs.getString("periodo_academico")
                );
                asignaciones.add(pm);
            }
        }
        return asignaciones;
    }
}
```

EstudianteMateriaDAO: Gestiona las matriculaciones de estudiantes

```
package dao;

import model.EstudianteMateria;
import java.sql.*;
import java.util.ArrayList;
```

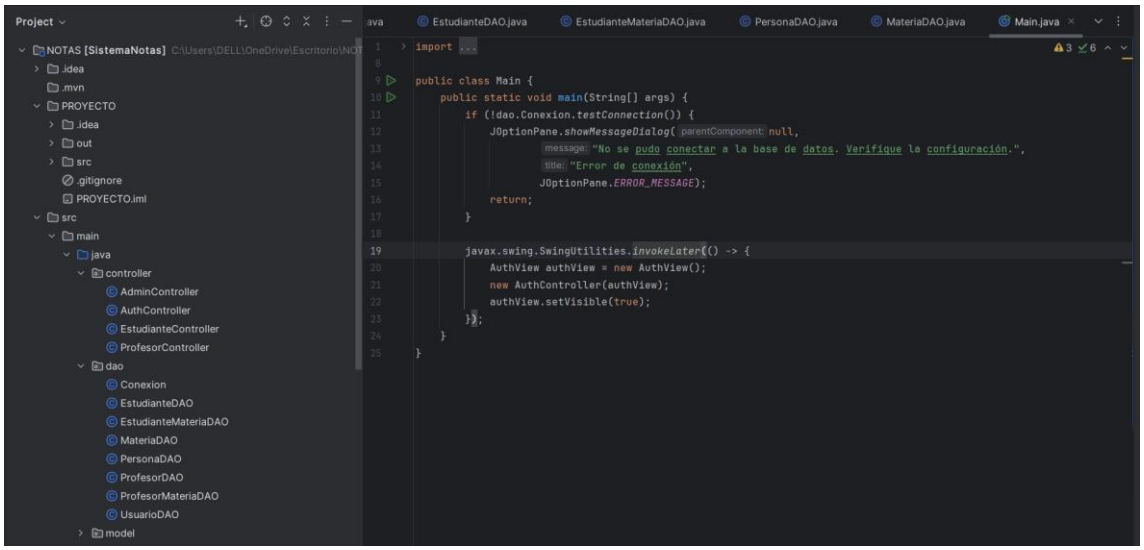
```
import java.util.List;

public class EstudianteMateriaDAO {

    public List<EstudianteMateria>
listarPorEstudiante(int idEstudiante) throws SQLException
{
        String sql = "SELECT id_materia,
periodo_academico FROM EstudianteMateria WHERE
id_estudiante = ?";
        List<EstudianteMateria> materias = new
ArrayList<>();

        try (Connection conn = Conexion.getConnection();
             PreparedStatement stmt =
conn.prepareStatement(sql)) {

            stmt.setInt(1, idEstudiante);
            ResultSet rs = stmt.executeQuery();

            while (rs.next()) {
                materias.add(new EstudianteMateria(
                        idEstudiante,
                        rs.getInt("id_materia"),
                        rs.getString("periodo_academico")
                ));
            }
        }
        return materias;
    }
    public boolean matricularEstudiante(int idEstudiante,
int idMateria, String periodoAcademico) throws
SQLException {
        String sql = "INSERT INTO EstudianteMateria
(id_estudiante, id_materia, periodo_academico) VALUES (?,
?, ?)";

        try (Connection conn = Conexion.getConnection();
             PreparedStatement stmt =
conn.prepareStatement(sql)) {

            stmt.setInt(1, idEstudiante);
            stmt.setInt(2, idMateria);
            stmt.setString(3, periodoAcademico);

            return stmt.executeUpdate() > 0;
        }
    }
```

```
    public boolean desmatricularEstudiante(int
idEstudiante, int idMateria, String periodoAcademico)
throws SQLException {
        String sql = "DELETE FROM EstudianteMateria WHERE
id_estudiante = ? AND id_materia = ? AND
periodo_academico = ?";

        try (Connection conn = Conexion.getConnection();
             PreparedStatement stmt =
conn.prepareStatement(sql)) {

            stmt.setInt(1, idEstudiante);
            stmt.setInt(2, idMateria);
            stmt.setString(3, periodoAcademico);

            return stmt.executeUpdate() > 0;
        }
    }

    public boolean estaMatriculado(int idEstudiante, int
idMateria, String periodoAcademico) throws SQLException {
        String sql = "SELECT 1 FROM EstudianteMateria
WHERE id_estudiante = ? AND id_materia = ? AND
periodo_academico = ?";

        try (Connection conn = Conexion.getConnection();
             PreparedStatement stmt =
conn.prepareStatement(sql)) {

            stmt.setInt(1, idEstudiante);
            stmt.setInt(2, idMateria);
            stmt.setString(3, periodoAcademico);

            try (ResultSet rs = stmt.executeQuery()) {
                return rs.next();
            }
        }
    }

    public List<EstudianteMateria> listarTodas() throws
SQLException {
        String sql = "SELECT id_estudiante, id_materia,
periodo_academico FROM EstudianteMateria";
        List<EstudianteMateria> asignaciones = new
ArrayList<>();

        try (Connection conn = Conexion.getConnection();
             PreparedStatement stmt =
conn.prepareStatement(sql);
             ResultSet rs = stmt.executeQuery()) {
```

```
            while (rs.next()) {
                asignaciones.add(new EstudianteMateria(
                        rs.getInt("id_estudiante"),
                        rs.getInt("id_materia"),
                        rs.getString("periodo_academico")
                ));
            }
        }
        return asignaciones;
    }

    public List<EstudianteMateria>
listarMateriasPorEstudiante(int idEstudiante) throws
SQLException {
        String sql = "SELECT id_materia,
periodo_academico FROM EstudianteMateria WHERE
id_estudiante = ?";
        List<EstudianteMateria> materias = new
ArrayList<>();

        try (Connection conn = Conexion.getConnection();
             PreparedStatement stmt =
conn.prepareStatement(sql)) {

            stmt.setInt(1, idEstudiante);
            ResultSet rs = stmt.executeQuery();

            while (rs.next()) {
                materias.add(new EstudianteMateria(
                        idEstudiante,
                        rs.getInt("id_materia"),
                        rs.getString("periodo_academico")
                ));
            }
        }
        return materias;
    }

    public List<EstudianteMateria>
listarEstudiantesPorMateria(int idMateria) throws
SQLException {
        String sql = "SELECT id_estudiante,
periodo_academico FROM EstudianteMateria WHERE id_materia
= ?";
        List<EstudianteMateria> estudiantes = new
ArrayList<>();

        try (Connection conn = Conexion.getConnection();
             PreparedStatement stmt =
conn.prepareStatement(sql)) {
```

```
            stmt.setInt(1, idMateria);
            ResultSet rs = stmt.executeQuery();

            while (rs.next()) {
                estudiantes.add(new EstudianteMateria(
                        rs.getInt("id_estudiante"),
                        idMateria,
                        rs.getString("periodo_academico")
                ));
            }
        }
        return estudiantes;
    }

    public List<EstudianteMateria>
listarPorPeriodo(String periodoAcademico) throws
SQLException {
        String sql = "SELECT id_estudiante, id_materia
FROM EstudianteMateria WHERE periodo_academico = ?";
        List<EstudianteMateria> asignaciones = new
ArrayList<>();

        try (Connection conn = Conexion.getConnection();
             PreparedStatement stmt =
conn.prepareStatement(sql)) {

            stmt.setString(1, periodoAcademico);
            ResultSet rs = stmt.executeQuery();

            while (rs.next()) {
                asignaciones.add(new EstudianteMateria(
                        rs.getInt("id_estudiante"),
                        rs.getInt("id_materia"),
                        periodoAcademico
                ));
            }
        }
        return asignaciones;
    }
}
```

Imagen 1: Presentación de las carpetas



El esquema de base de datos presenta una estructura bien normalizada que refleja las relaciones del mundo real en un sistema educativo:

sqlPersona (tabla padre) → Usuario, Estudiante, Profesor (herencia)

La tabla Persona actúa como una superclase que contiene la información común (nombres, apellidos, email, teléfono, tipo), mientras que las tablas Estudiante y Profesor extienden esta información con datos específicos como matrícula y especialidad respectivamente.

Las relaciones muchos-a-muchos entre profesores-materias y estudiantes-materias se resuelven mediante tablas intermedias (ProfesorMateria y EstudianteMateria), lo que permite un modelado flexible de las asignaciones académicas por períodos.

El sistema implementa restricciones de integridad referencial mediante claves foráneas con cascada en eliminación (ON DELETE CASCADE), asegurando que la eliminación de una persona automáticamente elimine sus registros asociados en las tablas dependientes.

Conectividad y Persistencia de Datos

Clase Conexion

La clase Conexion implementa el patrón Singleton implícito para gestionar las conexiones a la base de datos. Los aspectos más destacados incluyen:

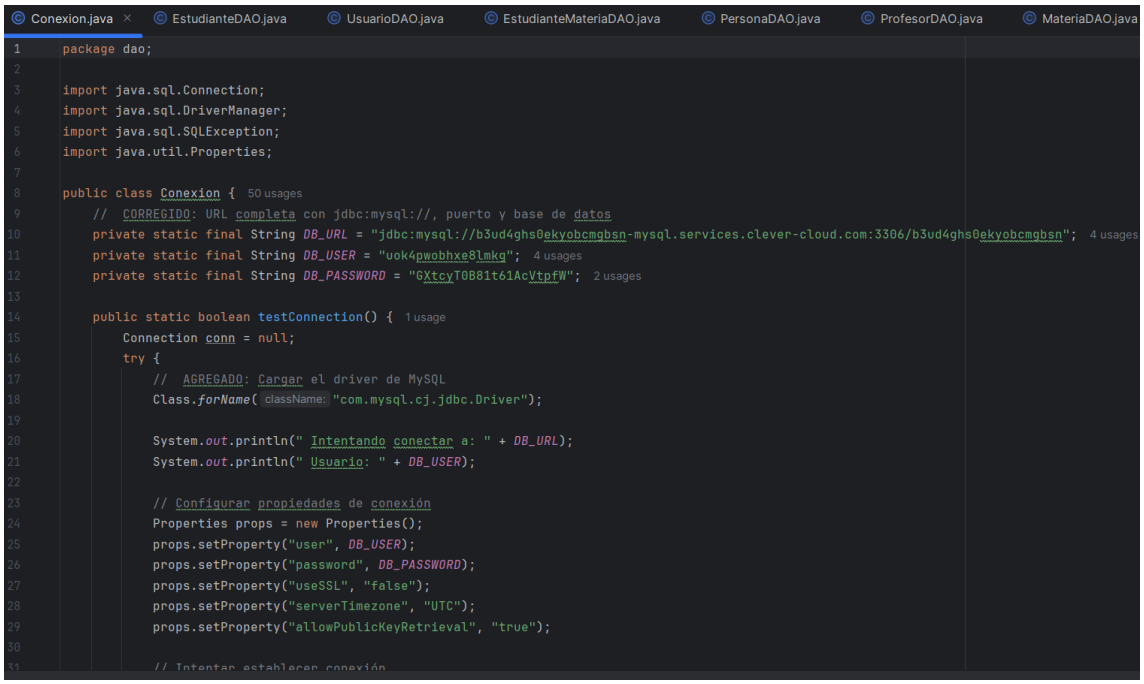
Configuración de Driver: Se carga explícitamente el driver MySQL (com.mysql.cj.jdbc.Driver)

Propiedades de Conexión: Configuración específica para trabajar con bases de datos en la nube

Manejo de Excepciones: Implementación robusta de manejo de errores con información detallada para debugging

Base de Datos en la Nube: Utilización de Clever Cloud como proveedor de base de datos MySQL

Imagen 2: Conexión



Durante el desarrollo se enfrentaron varios desafíos técnicos relacionados con la conectividad:

Configuración del Driver: Inicialmente hubo problemas con la carga del driver MySQL, resueltos mediante la carga explícita con Class.forName()

Parámetros de Conexión: La configuración de propiedades como useSSL=false, serverTimezone=UTC, y allowPublicKeyRetrieval=true fue crucial para establecer conexiones estables

Compatibilidad con IntelliJ IDEA: Se tuvieron que ajustar las configuraciones del IDE para trabajar correctamente con la base de datos remota

El AdminController demuestra una implementación completa de operaciones CRUD:

Create (Crear):

- Registro de nuevos usuarios con validación de datos
- Creación automática de registros en tablas relacionadas (Persona → Usuario)
- Asignación de materias a profesores y estudiantes

Read (Leer):

- Carga de datos en tablas con joins complejos
- Consultas que combinan información de múltiples entidades
- Actualización automática de interfaces gráficas

Update (Actualizar):

- Modificación de usuarios existentes con preservación de datos
- Actualización condicional de contraseñas
- Sincronización entre tablas relacionadas

Delete (Eliminar):

- Eliminación con confirmación del usuario
- Aprovechamiento de cascadas para mantener integridad referencial

La clase EstudianteDAO implementa funcionalidades específicas del dominio educativo:

- Gestión de Notas: Método actualizarNotas() que implementa lógica upsert (insertar o actualizar)

- Consultas Complejas: Joins entre múltiples tablas para obtener información completa del estudiante
- Manejo de Valores NULL: Tratamiento específico para notas no asignadas
- Matriculación: Gestión de inscripciones por períodos académicos

El desarrollo de la interfaz gráfica representó uno de los aspectos más desafiantes y gratificantes de nuestro proyecto. Decidimos utilizar Java Swing como framework para la construcción de las interfaces debido a su integración nativa con Java y la capacidad de crear aplicaciones de escritorio robustas. Durante el proceso de desarrollo, nos dimos cuenta de que crear una interfaz intuitiva y funcional requería no solo conocimientos técnicos, sino también una comprensión profunda de cómo los usuarios interactúan con los sistemas.

La arquitectura de nuestra interfaz gráfica se basa en un enfoque modular donde cada tipo de usuario tiene su propia ventana especializada. Esta decisión arquitectural surgió de la necesidad de proporcionar experiencias diferenciadas según el rol del usuario en el sistema educativo. Así, implementamos ventanas separadas para administradores, profesores y estudiantes, cada una extendiendo JFrame y proporcionando funcionalidades específicas para cada rol. Esta separación nos permitió personalizar completamente la experiencia de cada usuario sin crear interfaces sobrecargadas o confusas.

Para la estructura principal de las ventanas, empleamos diferentes gestores de diseño según las necesidades específicas de cada interfaz. En las ventanas principales utilizamos BorderLayout porque nos permitía dividir claramente las áreas de contenido,

colocando barras de herramientas en la parte superior, contenido principal en el centro y controles de acción en la parte inferior. Para los formularios más complejos, como el diálogo de registro de usuarios, implementamos GridBagLayout debido a su flexibilidad para manejar componentes de diferentes tamaños y alineaciones. Los controles simples como barras de botones los organizamos con FlowLayout por su simplicidad y comportamiento predecible.

Una de las características más importantes de nuestra interfaz administrativa es el uso de JTabbedPane para organizar las diferentes funcionalidades del sistema. Esta decisión de diseño surgió cuando nos dimos cuenta de que incluir todas las funciones administrativas en una sola ventana resultaba abrumador para el usuario. Las pestañas nos permitieron agrupar lógicamente las operaciones: una pestaña dedicada a la gestión de usuarios, otra para las asignaciones de profesores a materias, y una tercera para las matriculaciones de estudiantes. Cada pestaña funciona como un módulo independiente con sus propios controles y tablas de datos.

El componente central de nuestra aplicación son las tablas de datos, implementadas mediante JTable con modelos personalizados. Inicialmente tuvimos dificultades para entender cómo funcionaban los TableModel de Swing, pero una vez que comprendimos el patrón, pudimos crear tablas dinámicas que se actualizan automáticamente cuando los datos cambian en la base de datos. Por ejemplo, en la gestión de usuarios, nuestra tabla muestra columnas para ID, nombre de usuario, rol y nombre completo, y cada vez que se agrega, edita o elimina un usuario, la tabla se refresca automáticamente llamando a métodos como cargarUsuarios() desde el controlador.

Para manejar la selección de datos en formularios, implementamos JComboBox dinámicos que se pueblan directamente desde la base de datos. Esta fue una de las partes

más interesantes del desarrollo porque tuvimos que resolver cómo mostrar información legible al usuario mientras mantenemos los identificadores necesarios para las operaciones de base de datos. Nuestra solución fue crear cadenas con formato "ID - Nombre" que se muestran al usuario, pero luego extraemos el ID mediante parsing cuando necesitamos realizar operaciones. Este enfoque lo aplicamos tanto para la selección de profesores como para materias, y resultó ser muy efectivo para la usabilidad.

Los diálogos de confirmación y entrada de datos representaron otro aspecto fundamental de nuestra interfaz. Utilizamos JOptionPane para confirmaciones simples, especialmente para operaciones destructivas como eliminar usuarios o desasignar materias. Para operaciones más complejas como el registro de nuevos usuarios, creamos diálogos modales personalizados que incluyen múltiples campos de entrada, validación en tiempo real y selección de roles. Estos diálogos fueron particularmente desafiantes de implementar porque requerían coordinar la validación de datos, el manejo de errores y la actualización de la interfaz principal.

El manejo de eventos en nuestra aplicación aprovecha las expresiones lambda de Java 8, lo que nos permitió escribir código más limpio y legible. En lugar de crear clases anónimas extensas para cada ActionListener, pudimos usar referencias a métodos como this::agregarUsuario o this::editarUsuario. Esta aproximación no solo reduce el código repetitivo, sino que también hace más evidente la conexión entre los elementos de la interfaz y la lógica de negocio en el controlador.

Un aspecto crucial que implementamos fue el sistema de retroalimentación visual para el usuario. Cada operación en el sistema proporciona feedback inmediato, ya sea confirmando el éxito de una operación o informando sobre errores específicos. Los mensajes de éxito aparecen cuando se completan operaciones como "Usuario registrado

exitosamente", mientras que los errores proporcionan información detallada sobre qué salió mal, como "Error de base de datos" seguido del mensaje específico de la excepción. Esta retroalimentación fue esencial para crear una experiencia de usuario confiable.

La actualización automática de la interfaz después de cada operación CRUD fue uno de los aspectos más importantes del diseño. Cada vez que se realiza una operación exitosa, como agregar un nuevo usuario o asignar una materia a un profesor, la interfaz se actualiza automáticamente para reflejar los cambios. Esto se logra mediante llamadas a métodos de carga como cargarUsuarios(), cargarAsignaciones() o cargarAsignacionesEstudiantes() que refrescan las tablas correspondientes con los datos más recientes de la base de datos.

La navegación entre diferentes partes del sistema se diseñó teniendo en cuenta los roles de usuario y la seguridad. Cuando un usuario inicia sesión, el sistema determina su rol y presenta únicamente las opciones relevantes para ese tipo de usuario. Los administradores tienen acceso completo a todas las funcionalidades, incluyendo gestión de usuarios y asignaciones. Los profesores ven interfaces enfocadas en la gestión de notas y consulta de estudiantes en sus materias. Los estudiantes acceden a vistas simplificadas donde pueden consultar sus calificaciones y materias inscritas.

Para el cierre de sesión y transición entre ventanas, implementamos un sistema que cierra apropiadamente la ventana actual y abre la ventana de login, permitiendo que otros usuarios accedan al sistema sin necesidad de reiniciar la aplicación. Este flujo se maneja mediante la disposición de la ventana actual y la creación de nuevas instancias de la vista de autenticación junto con su controlador correspondiente.

Durante el desarrollo de las interfaces, también tuvimos que considerar aspectos de usabilidad como el redimensionamiento de ventanas y la adaptabilidad de los componentes. Las tablas se configuraron con columnas redimensionables y barras de desplazamiento para manejar grandes volúmenes de datos sin comprometer la usabilidad. Los formularios mantienen proporciones apropiadas incluso cuando se redimensiona la ventana, y los botones conservan tamaños consistentes que facilitan la interacción.

El diseño específico para cada rol requirió un análisis cuidadoso de las necesidades de cada tipo de usuario. La interfaz administrativa incluye controles completos para gestión de usuarios, creación y eliminación de cuentas, asignación de materias y matriculación de estudiantes. La interfaz de profesor se enfoca en la visualización de materias asignadas y la gestión de calificaciones de estudiantes. La interfaz de estudiante proporciona una vista simplificada centrada en la consulta de calificaciones personales y el historial académico.

Una de las lecciones más importantes que aprendimos durante el desarrollo de la interfaz gráfica fue la importancia de la consistencia en el diseño. Mantuvimos patrones consistentes para operaciones similares, como usar siempre diálogos de confirmación para eliminaciones o proporcionar mensajes de retroalimentación después de cada acción. Esta consistencia no solo facilitó el desarrollo, sino que también mejoró significativamente la experiencia del usuario al hacer el sistema más predecible e intuitivo de usar.

ANEXOS

Imagen 3: Area de carpetas creadas y desarrolladas

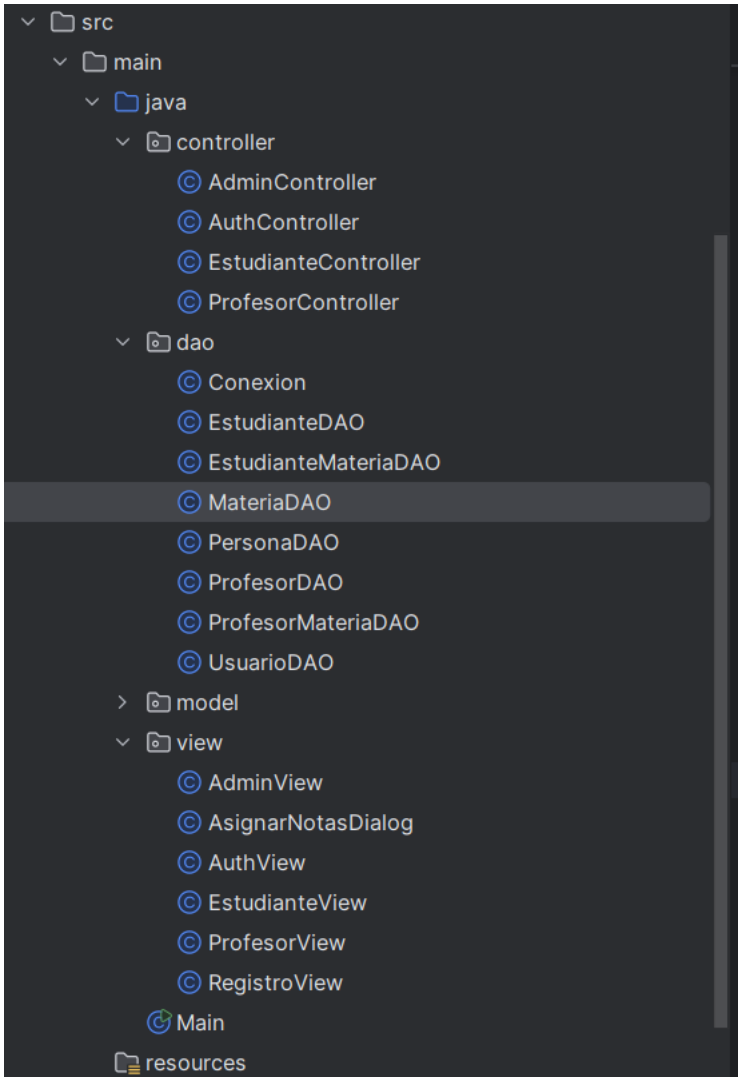


Imagen 4: Sistema de ingreso al sistema

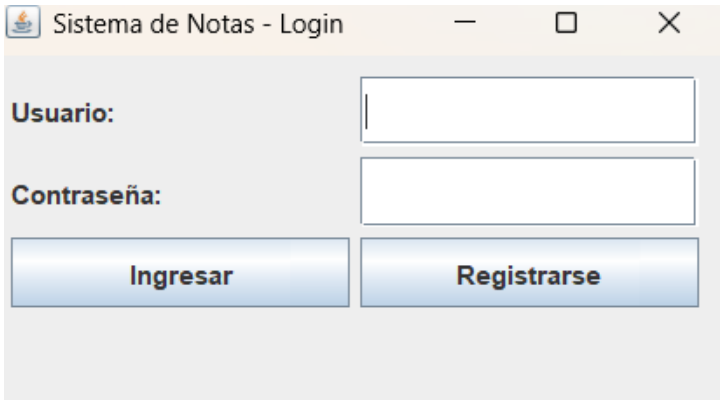


Imagen 5: Sistema de registro

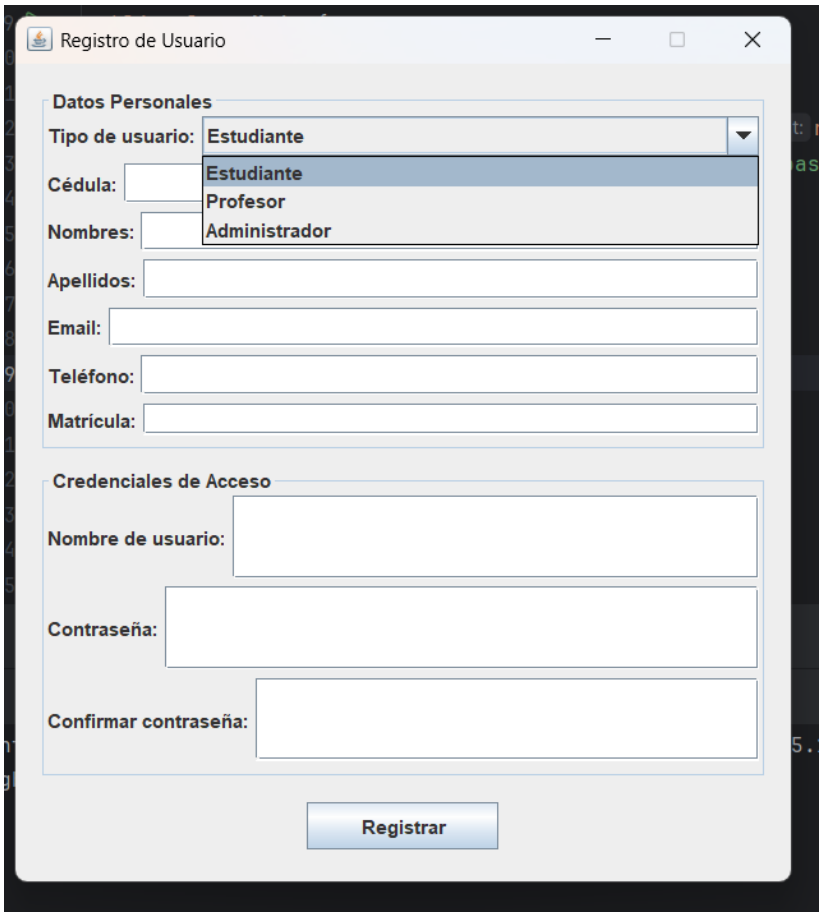


Imagen 6:Sistema de Administrador

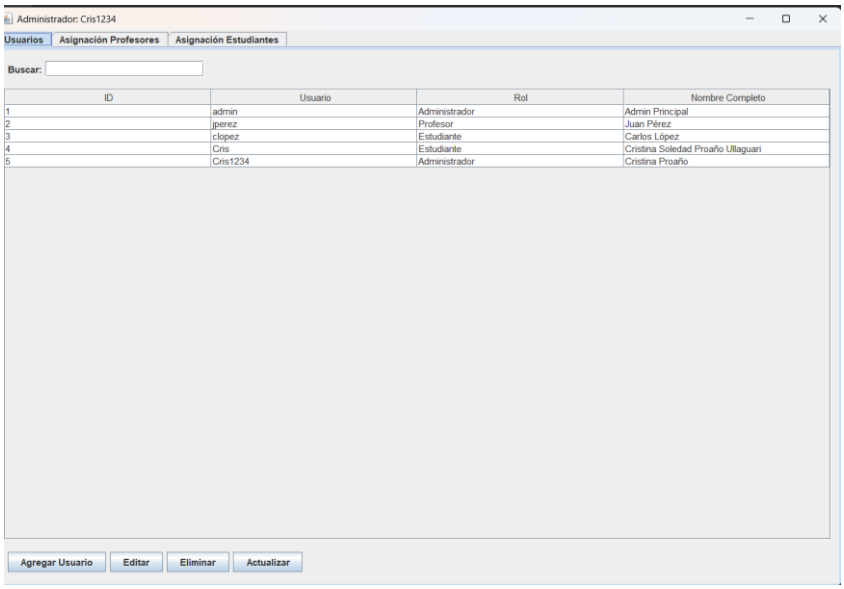


Imagen 7: Area de carpetas creadas y desarrollo de MAIN

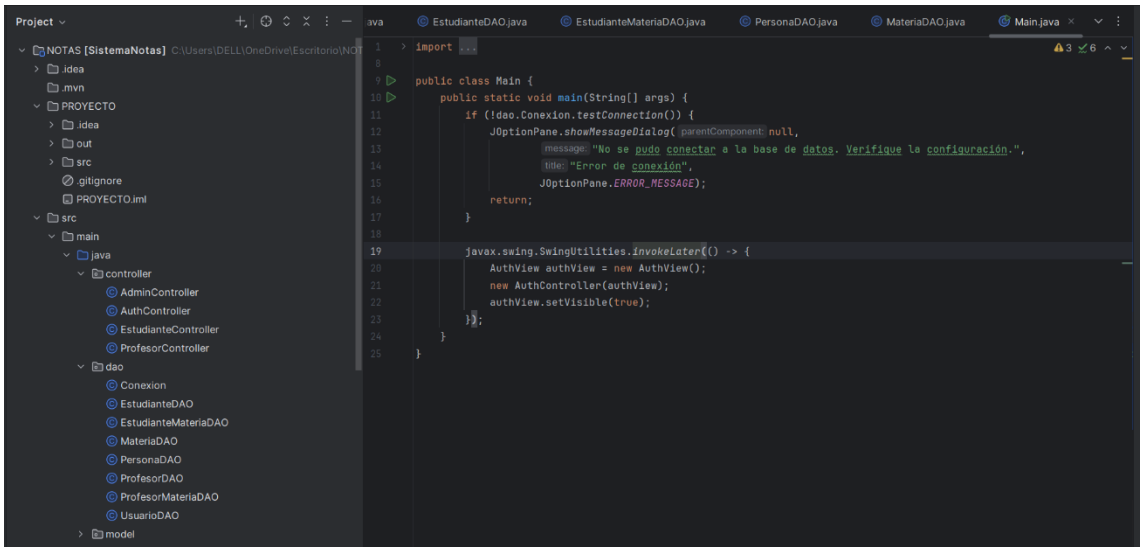


Imagen 8: CLEVER CLOUD

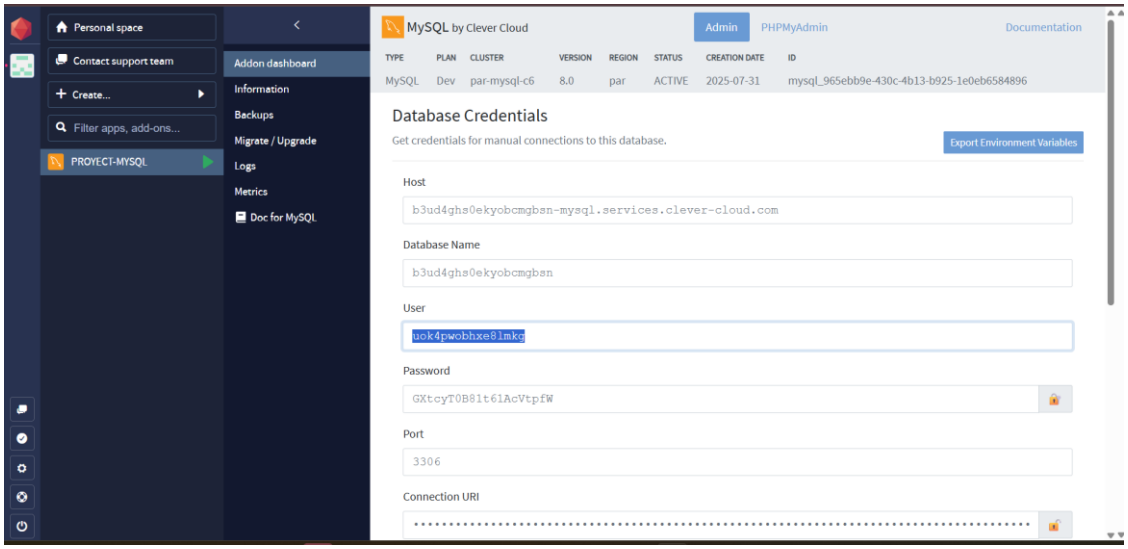
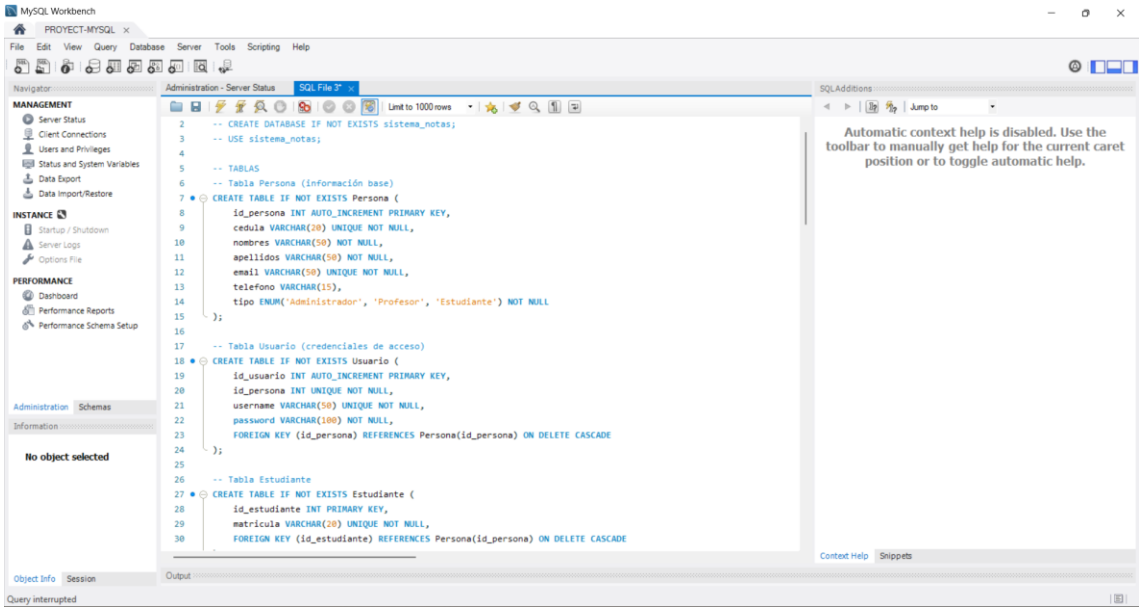


Imagen 9: MYSQL WORKBENCH



CONCLUSIONES

La construcción del Sistema de Gestión de Notas Académicas ha sido una experiencia formativa integral que me permitió aplicar de manera concreta los fundamentos de la Programación Orientada a Objetos (POO) en el desarrollo de un sistema funcional con utilidad en un contexto académico real. A lo largo del proyecto, pude evidenciar cómo los pilares de la POO como la abstracción, encapsulamiento, herencia y polimorfismo no solo son conceptos teóricos, sino herramientas potentes para diseñar soluciones robustas, escalables y mantenibles. Cada clase, objeto y relación implementada en el sistema respondía a una necesidad real del proceso de gestión académica, lo cual fortaleció mi comprensión sobre la arquitectura lógica detrás de los sistemas modernos.

Uno de los aprendizajes más significativos fue la integración de múltiples tecnologías, desde la estructura de una base de datos bien normalizada hasta la creación de interfaces gráficas intuitivas para el usuario. Esta sinergia entre la lógica del backend y la experiencia del usuario en el frontend me permitió consolidar mi conocimiento en diseño modular, separación de responsabilidades y reutilización del código. Además, el proceso de prueba, depuración y mejora constante me hizo tomar conciencia de la importancia del ciclo de vida del software y de las buenas prácticas de desarrollo.

A pesar de que el sistema cumple satisfactoriamente con los requisitos establecidos, soy consciente de que todo proyecto de software puede evolucionar y adaptarse a nuevas necesidades. Por esta razón, considero que hay varias mejoras que podrían potenciar aún más su funcionalidad. Por ejemplo, la incorporación de un área de reportes que permita generar informes académicos en formato PDF resultaría de gran valor para docentes y autoridades educativas. Asimismo, la implementación de un sistema de notificaciones automáticas ayudaría a mantener informados a los usuarios sobre fechas importantes como exámenes, cierre de calificaciones o inscripciones.

Otro paso importante sería llevar el sistema hacia una interfaz web moderna, mejorando así la accesibilidad y usabilidad desde distintos dispositivos. Esto podría complementarse con la creación de una API REST que exponga los servicios principales del sistema, permitiendo su integración con otras plataformas académicas o institucionales. Finalmente, incluir un mecanismo de backup automático garantizaría la seguridad y recuperación de la información ante posibles fallos.

En conclusión, este proyecto no solo me permitió demostrar lo aprendido en la asignatura de POO, sino que también me impulsó a pensar como un desarrollador de software profesional, enfrentando retos reales y proponiendo soluciones prácticas. Me voy con la convicción de que la programación orientada a objetos no es solo una técnica de codificación, sino una filosofía que guía la manera en que concebimos, diseñamos y construimos sistemas útiles y sostenibles.