Universidad Finis Terrae
Facultad de Ingeniería Civil
Bases de datos
Segundo semestre 2025

Profesor: Fernando Rafael San Martín Woerne
Ayudante: Juan Eulogio Rivera Vásquez

Sistema de Gestión de Información Escolar

Integrantes: Millaray Olivares
Valentín Latú
Matina Vargas
Matías Morales

Fecha de entrega: 21 de noviembre de 2025

Índice

Introducción

Para la séptima y última semana de desarrollo del proyecto, se tomó el enfoque de documentar y preparar todo para la entrega final. Luego de completar la implementación y validación de las consultas SQL esenciales o CRUD, y el desarrollo de la interfaz durante la semana 6, en esta etapa se presenta la solución completa del sistema de gestión de información escolar, teniendo como propósito central de esta semana tener los entregables definitivos según lo estipulado.

El primer punto es la preparación de la aplicación funcional, lo que implica organizar y pulir el código de la aplicación, para poder asegurar que la base de datos esté poblada con los datos de prueba representativos y funcionales, además de redactar un manual claro con las instrucciones de instalación y ejecución. Por otro lado, tenemos la redacción del documento, donde este debe compilar de manera estructurada y coherente todo el trabajo realizado durante el semestre, esto incluirá la descripción del problema, el modelo físico normalizado , los scripts DDL de la estructura , el detalle de las consultas DML utilizadas , la descripción de la aplicación desarrollada y los resultados de las pruebas.

Finalmente, luego de haber realizado cada punto, se aplicará una revisión sobre cada componente del proyecto, con el objetivo de asegurar la coherencia total entre la documentación y el producto software, corrigiendo errores o cualquier inconsistencia, refinándola y preparando una solución sólida y completa para la entrega final.

Desarrollo

Descripción detallada del problema y requisitos.

Caso de estudio: Sistema de Gestión de Información Escolar
Los colegios son organizaciones que almacenan grandes cantidades de información sobre estudiantes, docentes, cursos, notas, certificados, comunicaciones o avisos y asistencia, entre otras. Todo esto, por lo general, es almacenado en planillas o registros manuales como el tan famoso libro de clase, los cuales son propensos a provocar errores al registrar información o generar duplicidad de datos en el registro académico. Esta situación dificulta llevar un historial claro sobre el rendimiento de los estudiantes y complica la generación de reportes para directivos o autoridades. Además, otro punto que siempre suele ser limitado o tardío es la comunicación con los apoderados, lo que genera una desconfianza en el sistema educativo. Una base de datos centralizada permitiría integrar toda esta información en un sistema único, reduciendo errores para agilizar procesos administrativos y académicos, ofreciendo herramientas de seguimiento más completas para mejorar la gestión escolar.

Los requisitos de funcionalidades específicas que el sistema debe realizar son:

- Gestión de usuarios: Es el registro, modificación y eliminación de la información y perfiles de alumnos, profesores y apoderados.

- Gestión académica: Es un control integral de los procesos educativos como notas o asistencia diaria.

- Comunicación y participación: Consta de una plataforma de interacción que permite el envío y consulta de avisos o mensajes entre profesores y alumnos o apoderados.

- Base de datos y mantenimiento: Es el aseguramiento de la integridad operativa del sistema mediante respaldos, restauración de datos, control de duplicados y cambios.

- Información y reportes: Funcionalidad para la búsqueda de usuarios por rut y la generación de informes detallados sobre la información de los alumnos.

Modelo conceptual ER.

El modelo entidad-relación (ER) diseñado para el sistema de gestión de información escolar, este diagrama representa la estructura lógica de la base de datos, identificando las entidades principales como Alumno, Profesor, Apoderado, Curso y Notas, junto con sus atributos y las relaciones necesarias para los requisitos funcionales definidos. Este modelo sirve como plano conceptual para garantizar la integridad de los datos antes del traspaso a su implementación física.
Ya que el tamaño del diagrama es grande, se agregó en el Anexo 1.

Modelo físico normalizado hasta 3FN.

En esta sección, se presenta la transformación del modelo conceptual hacia el modelo relacional definitivo del Sistema de Gestión Escolar. El modelo incluye la definición de claves primarias y foráneas, las reglas de normalización hasta tercera forma normal (3FN) y las restricciones de integridad.

A partir del análisis del caso escolar, se identificaron las entidades necesarias para representar los procesos académicos y administrativos del establecimiento. La tabla a continuación resume las entidades, atributos principales, clave primaria y claves foráneas resultantes:

Tabla	Claves primarias	Claves foráneas	Atributos principales
Alumno	id_alumno	id_apoderado → Apoderado(id_apoderado)id_curso → Curso(id_curso)	rut, nombre, apellido, correo, fecha_nacimiento, contraseña
Apoderado	id_apoderado	—	rut, nombre, apellido, correo, contraseña, fecha_nacimiento, telefono
Profesor	id_profesor	—	rut, nombre, apellido, correo, contraseña, fecha_nacimiento, telefono
Curso	id_curso	id_profesor_jefe → Profesor(id_profesor)	nombre
Asignatura	id_asignatura	id_profesor_jefe → Profesor(id_profesor)	nombre_asignatura
Nota	id_nota	id_alumno → Alumno(id_alumno)id_asignatura → Asignatura(id_asignatura)	fecha, nota
Asistencia	id_asistencia	id_alumno → Alumno(id_alumno)	fecha, presente
Comunicación	id_comunicacion	id_profesor → Profesor(id_profesor)id_alumno → Alumno(id_alumno)	fecha, mensaje

Tabla (1): Tablas y sus atributos/claves.

Primera Forma Normal (1FN)

La 1FN exige que cada atributo almacene un único valor atómico. En nuestro modelo final, todos los atributos cumplen esta condición: los teléfonos, correos y contraseñas son valores únicos por registro, las notas y fechas se registran en columnas atómicas y no existen listas ni campos multivaluados. Por ejemplo, la tabla Nota almacena un único valor nota por fila; la tabla Asistencia almacena un único valor booleano presente por registro.

Segunda Forma Normal (2FN)

La 2FN exige que, además de la 1FN, los atributos no clave dependan completamente de la clave primaria y que no existan dependencias parciales respecto a claves compuestas. En nuestro diseño, todas las tablas principales utilizan claves primarias simples (p. ej. id_alumno, id_curso), y los atributos no clave como correo, nombre o fecha_nacimiento dependen íntegramente de la PK correspondiente. Por ejemplo, en la tabla Nota los campos fecha y nota dependen de id_nota, y las referencias a alumno o asignatura se gestionan mediante claves foráneas (id_alumno, id_asignatura).

Tercera Forma Normal (3FN)

La 3FN exige que no existan dependencias transitivas entre atributos no clave. En nuestro modelo, la información como nombre del curso o los datos del profesor jefe se almacenan en la tabla Curso y no se repiten en Alumno o Nota. Por ejemplo, si cambiara el nombre del curso o el profesor_jefe, la actualización ocurre en una única fila de la tabla Curso, evitando inconsistencias y actualizaciones múltiples en tablas dependientes. Así, el modelo final cumple 3FN.

Implementación de la Base de Datos

Se implementó el modelo físico del sistema escolar dentro del gestor SQLite. El propósito fue materializar la estructura relacional diseñada, incorporando tablas, claves primarias y foráneas, restricciones, vistas e índices, junto con una carga inicial de datos de prueba que permitiera validar la integridad del sistema.

Creación de tablas, vistas, índices y restricciones en SQL:

Las tablas se crearon respetando la normalización hasta 3FN, definiendo claves primarias, claves foráneas, y restricciones como NOT NULL, UNIQUE y CHECK. Adicionalmente, se incorporaron índices para optimizar consultas frecuentes y se generó una vista que facilita el acceso a la información combinada entre alumnos y cursos.
Las tablas creadas fueron:

- apoderado.
- alumno.
- profesor.

- curso.
- asignatura.
- nota.
- asistencia.
- comunicación.

Además se implementaron:

- Índices: idx_alumno_rut, idx_profesor_rut.
- Vista: vista_alumno_detalle .

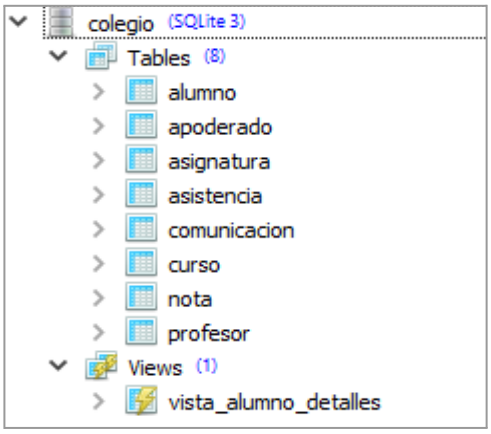


Figura 1: Estructura de tablas, vista e índices generados en SQLiteStudio.

Scripts DDL para la estructura completa:

A continuación, se presenta el script DDL utilizado para la creación de la estructura completa de la base de datos, incluyendo tablas, claves primarias, claves foráneas, restricciones de dominio, índices y la vista principal del sistema.
El diseño contempla todos los atributos definidos en el modelo lógico final, tales como contraseña, fecha_nacimiento y teléfono para los usuarios del sistema (alumnos, apoderados y profesores), junto a las relaciones necesarias para asegurar la integridad de los datos.

Para mantener la claridad del documento, el código completo se incluye en el Anexo 2, donde se detalla la creación de cada entidad según el modelo físico normalizado hasta Tercera Forma Normal (3FN).

Carga inicial de datos de prueba (DML):

Para validar la integridad y correcto funcionamiento del modelo implementado, se realizó una carga inicial de datos mediante sentencias DML (INSERT INTO).
Los datos insertados corresponden a registros representativos del contexto escolar, incluyendo:

- apoderados con credenciales.
- profesores con información completa.

- cursos con profesor jefe asignado.
- asignaturas vinculadas a cada profesor.
- alumnos asociados a apoderados y cursos.
- notas registradas por asignatura.
- asistencias por fecha.
- comunicaciones entre profesores y alumnos.

El proceso de poblamiento se realizó respetando el orden impuesto por las claves foráneas:

1. apoderado y profesor.
2. curso.
3. asignatura.
4. alumno.
5. nota.
6. asistencia.
7. comunicación.

Esto permitió verificar:

- Integridad referencial, asegurada por las FK.
- Cumplimiento de restricciones de dominio, como la condición CHECK del rango de notas.
- Unicidad de los campos rut y correo.
- Consistencia lógica entre entidades relacionadas.

Una vez cargados los datos, se comprobó que la estructura diseñada soporta correctamente todos los casos de uso del Sistema de Gestión Escolar, sirviendo como base funcional para las operaciones CRUD desarrolladas en Streamlit.

Las tablas generadas con los datos de prueba se presentan en el orden de ejecución del script DML. Las figuras completas (4 a 10), correspondientes al contenido final de cada tabla, se incluyen en el Anexo 3.

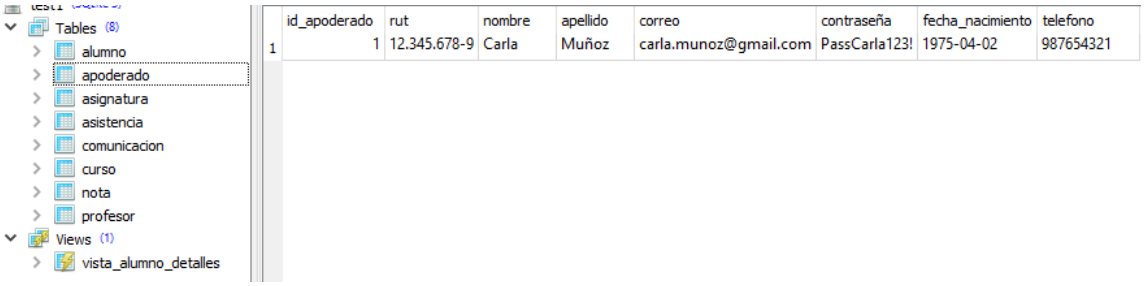


Figura 2: Tabla de apoderado resultante.

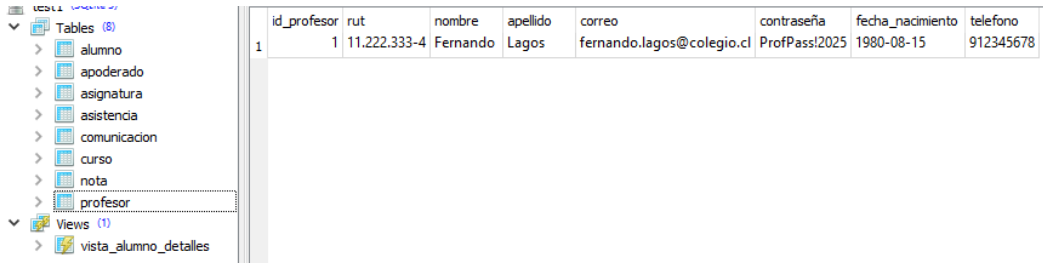


Figura 3: Tabla de profesor resultante.

Desarrollo de Funcionalidades Básicas

En esta etapa se implementaron las operaciones SQL esenciales (CRUD) y se desarrolló una interfaz mínima viable utilizando Streamlit para interactuar con la base de datos de forma gráfica.

Implementación de consultas de SQL CRUD:

Se implementaron las 4 operaciones CRUD sobre las entidades principales (alumno, curso, apoderado, notas, asistencia, comunicaciones). Estas consultas fueron implementadas y probadas en SQLiteStudio para verificar la correcta interacción entre las tablas creadas, las claves foráneas y las restricciones de integridad.
Las consultas hechas fueron:

- INSERT
- SELECT con JOIN
- UPDATE
- DELETE

Figura 11: Alumno insertado con sus parámetros en tabla.

El código SQL completo utilizado para cada operación CRUD se incluye en el Anexo 4.

Desarrollo de la interfaz mínima viable (Streamlit):

Para facilitar la interacción con la base de datos del Sistema de Gestión Escolar, se desarrolló una interfaz mínima viable utilizando Streamlit. Esta interfaz permite ejecutar operaciones CRUD sin necesidad de escribir consultas SQL manualmente, proporcionando una experiencia más intuitiva para los usuarios finales.

La aplicación establece la conexión con la base de datos mediante el módulo sqlite3, habilitando el uso de PRAGMA foreign_keys = ON para asegurar que las restricciones de integridad referencial se mantengan durante todas las operaciones realizadas.

La interfaz incorpora un módulo de autenticación que permite el ingreso de alumnos, profesores y apoderados mediante RUT y contraseña, validando credenciales almacenadas en la base de datos.

Elementos principales de la interfaz:

La interfaz está compuesta por formularios y controles interactivos que permiten:

- Crear nuevos registros mediante campos de entrada (INSERT).
- Consultar información de todas las tablas del sistema, incluyendo alumnos, apoderados, profesores, cursos, asignaturas, notas, asistencias y comunicaciones (SELECT).
- Editar información existente a través de formularios precargados (UPDATE).
- Eliminar registros mediante un mecanismo de confirmación para evitar errores (DELETE).

Además, la interfaz aplica validaciones esenciales para asegurar la calidad de los datos, incluyendo verificación de formato de RUT, correo electrónico, teléfono, fechas en formato DD/MM/YYYY, obligatoriedad de contraseñas y validación de claves foráneas antes de cada operación. Cada acción que ejecuta el usuario genera una sentencia SQL interna que es enviada a SQLite de manera automática.

Evidencia visual de la interfaz:

A continuación, se presenta una captura representativa del diseño y estructura general de la interfaz:

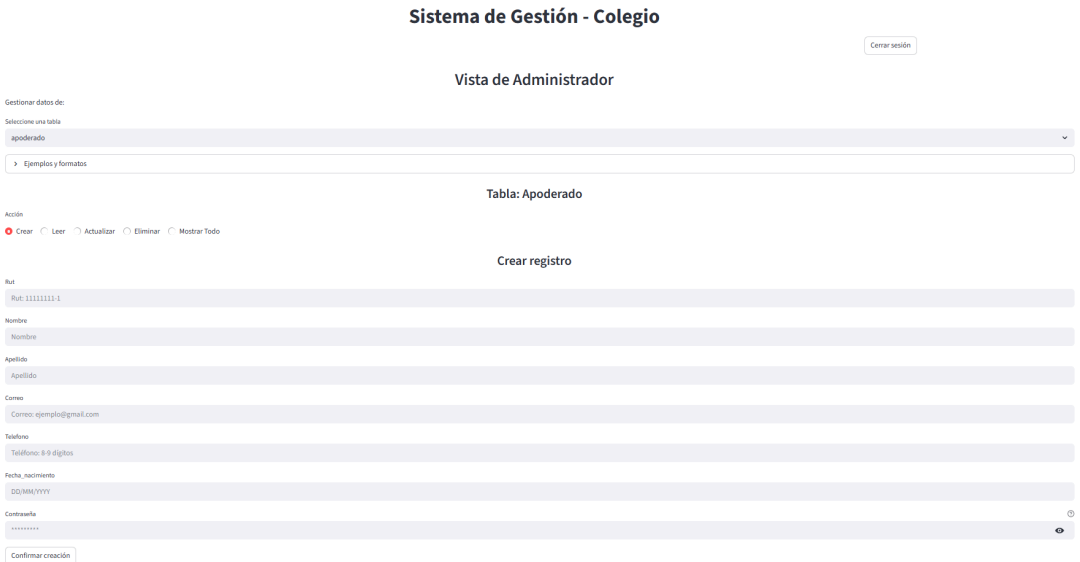


Figura 12: Interfaz en Streamlit con opciones CRUD.

Código de la interfaz:
Debido a su extensión, el código completo de la interfaz se encuentra incluido en el Anexo 5.

Pruebas realizadas y resultados obtenidos

Lo primero es hacer login con los datos de administrador, modificables en python, en este caso, usuario es “admin” y contraseña es “12345”.

Figura 13: Login de admin.

Al iniciar sesión como administrador, tenemos acceso completo a usar CRUD con cada entidad.

Figura 14: Ejemplos y formatos de cómo rellenar.

A continuación se inicia el “CRUD”:

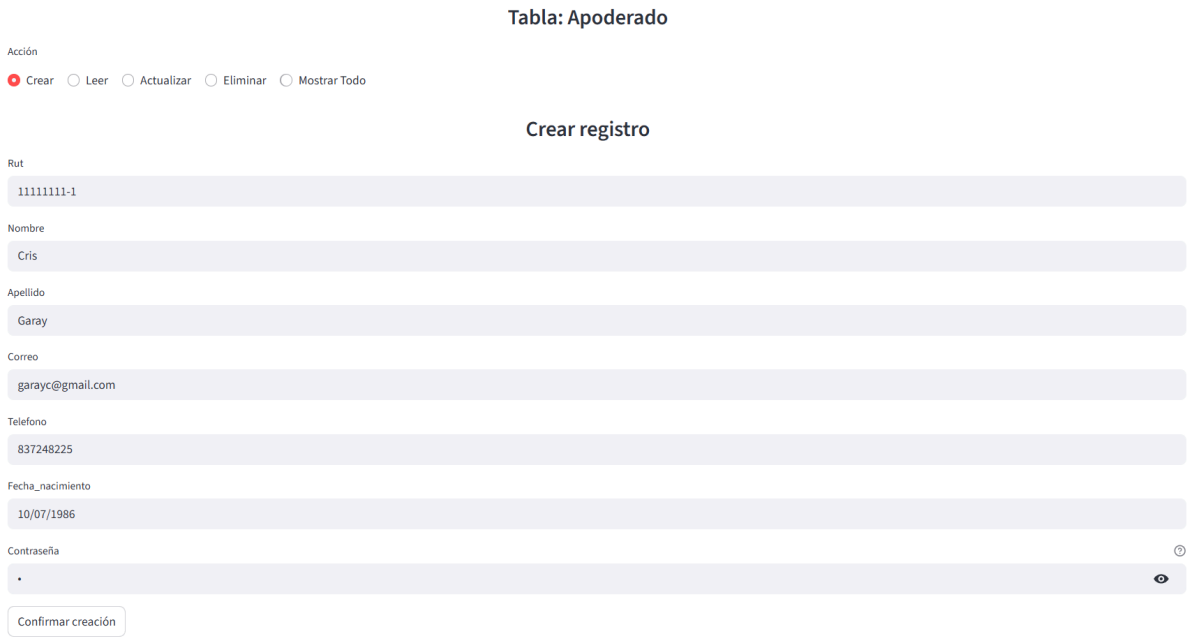


Figura 15: Datos de prueba para rellenar en un apoderado.

	id_apoderado	rut	nombre	apellido	correo	contraseña	fecha_nacimiento	telefono
0	1	12.345.678-9	Carla	Muñoz	carla.munoz@gmail.com	PassCarla123!	02/04/1975	987654321
1	2	11111111-1	Cris	Garay	garayc@gmail.com	1		837248225

Figura 16: Comprobación visual de que fue añadido con éxito.

	id_apoderado	rut	nombre	apellido	correo	contraseña	fecha_nacimiento	telefono
0	2	11111111-1	Cris	Garay	garayc@gmail.com	1		837248225

Figura 17: Actualización de datos.

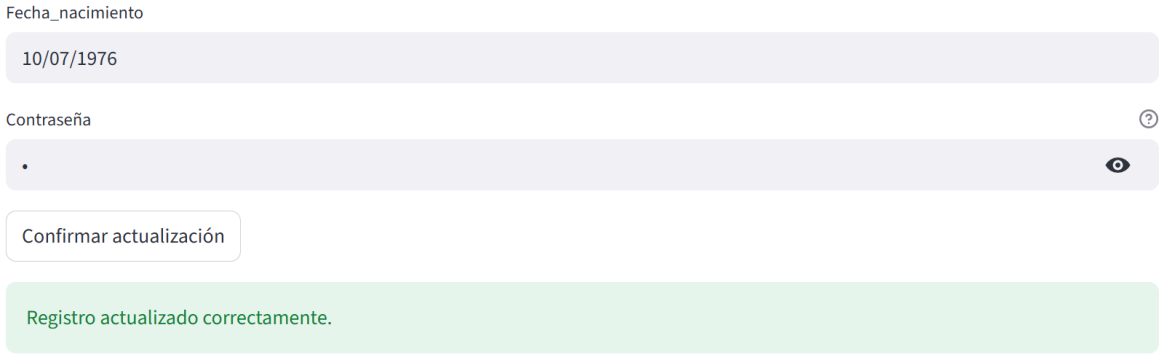


Figura 18: Cambio de fecha de nacimiento.

Figura 19: Comprobación visual de que fue actualizado con éxito.

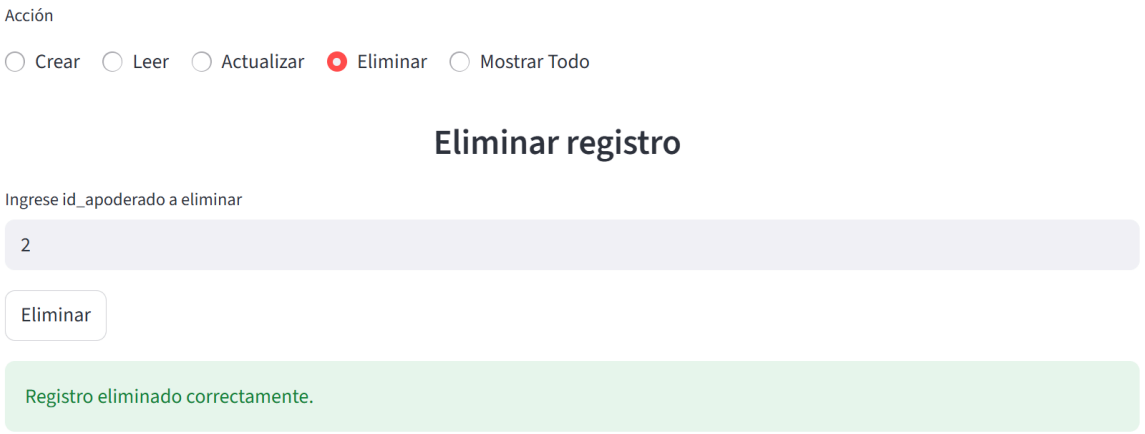


Figura 20: Eliminación del registro creado.

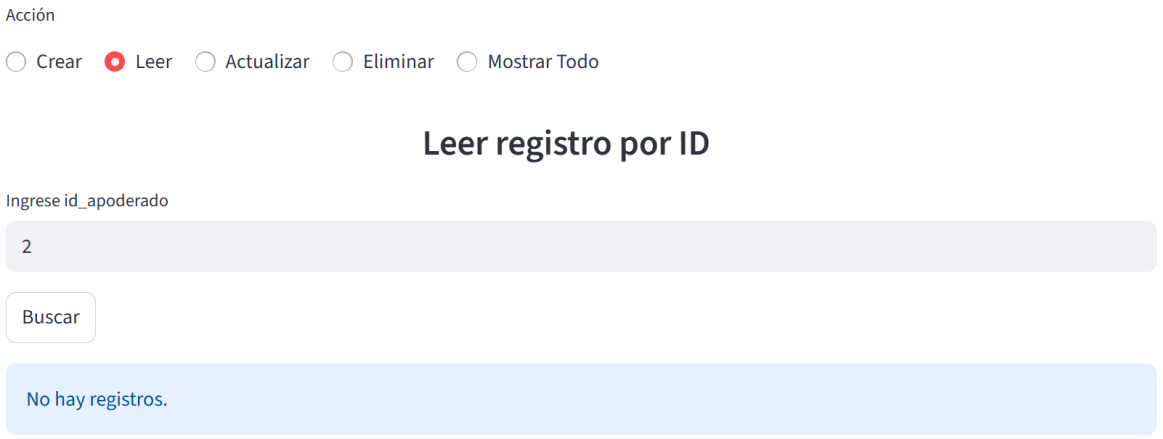


Figura 21: Ya no se encuentra en los registros.

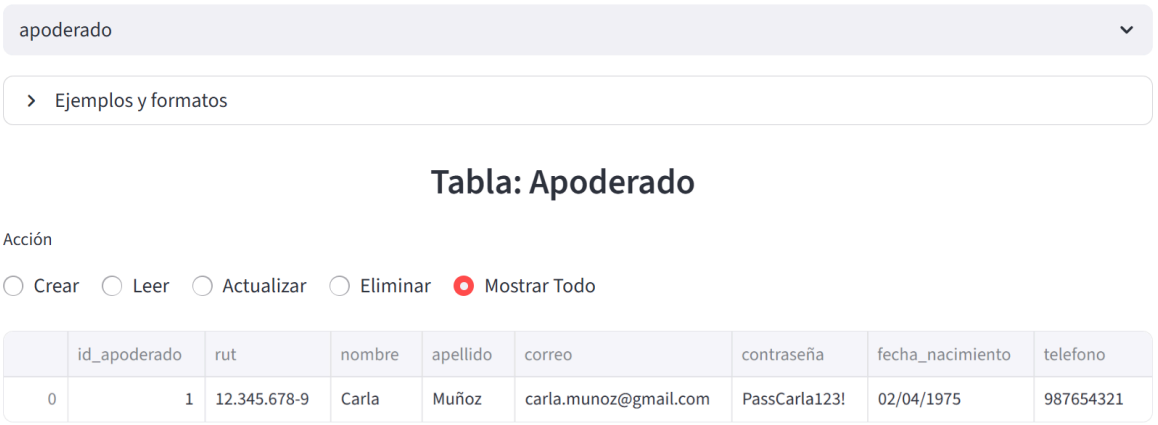


Figura 22: Comprobación visual de que fue eliminado con éxito.

Vista de los apoderados en la interfaz:

Figura 23: Login de apoderada Carla Muñoz.

Figura 24: Vista de la apoderada Carla Muñoz.

Vista de los alumnos en la interfaz:

Figura 25: Login de la alumna María Vargas.

Figura 26: Vista de la alumna Maria Vargas.

Vista de los profesores en la interfaz:

Figura 27: Login del profesor Fernando Lagos.

Figura 28: Vista del profesor Fernando Lagos.

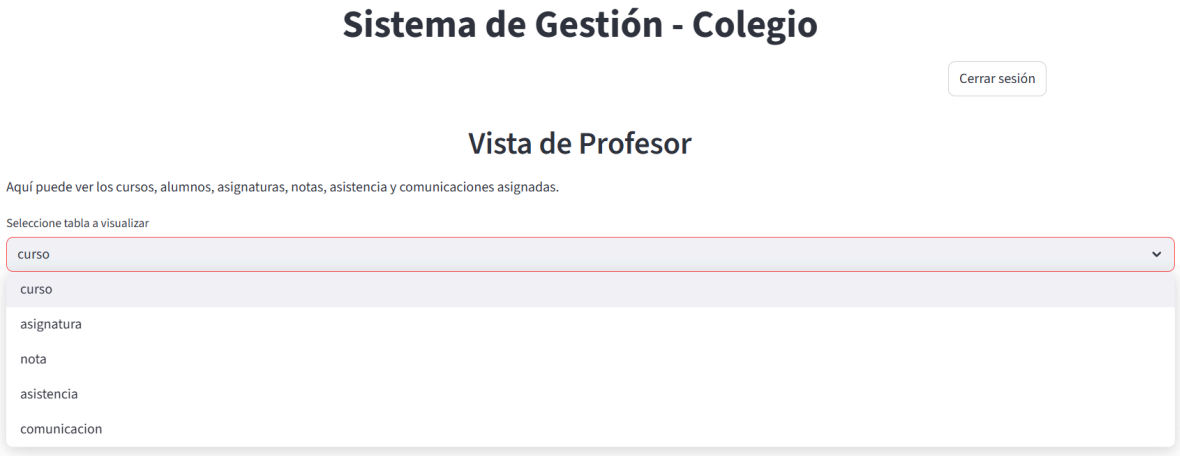


Figura 29: Vista de opciones múltiples del profesor Fernando Lagos.

Trabajo futuro

El sistema desarrollado constituye una base sólida para la gestión escolar; sin embargo, existen diversas mejoras que podrían implementarse en futuras versiones. Entre ellas se consideran:

- Mejorar el módulo de autenticación, integrando funciones como recuperación de contraseña, cifrado robusto y registro de usuarios.
- Desarrollar reportes automáticos en formatos PDF o Excel, que incluyan estadísticas de asistencia, calificaciones y rendimiento por curso.
- Añadir un panel estadístico con gráficos interactivos que muestren tendencias relevantes dentro del establecimiento.
- Implementar un sistema de notificaciones automáticas para informar a apoderados y profesores sobre eventos importantes (inasistencias, notas bajas, mensajes internos, etc.).
- Expandir el sistema hacia una plataforma web completa, con un backend más robusto y una interfaz moderna.
- Integrar un módulo de matrículas y gestión académica anual, que permita administrar promociones, historial académico y cambios de curso.

Y todas estas mejoras permitirían aumentar las funcionalidades del sistema, ampliabilidad y utilidad en el entorno escolar.

Conclusión

El desarrollo del sistema de gestión de información escolar se completó de forma correcta con la materialización de una base de datos relacional robusta y funcional, logrando el objetivo central de solucionar la ineficiencia, duplicidad de datos y la limitada accesibilidad a la gestión mediante planillas y registros manuales.

El proceso de desarrollo siguió una metodología estructurada, comenzando con la definición detallada del problema y la recopilación de requisitos necesarios para cada funcionalidad, estos se plasmaron en un modelo Entidad-Relación. El diseño consta de una solidez, la cual se confirmó en la etapa donde se realiza la transformación del modelo conceptual a un modelo relacional, aplicando una normalización de hasta tercera forma normal 3FN, donde se aseguró una estructura optimizada, libre de redundancias y con dependencia funcional completa, concluyendo con las bases de una plataforma de datos coherente. Luego se implementó en SQL, permitiendo la creación del modelo físico a través de script DDL, donde la correcta definición de claves primarias, foráneas y restricciones de dominio, como la validación del rango de notas, es crucial para conseguir la integridad referencial y de dominio. Estos resultados se complementaron con el desarrollo de las operaciones esenciales DML, incluyendo las funcionalidades CRUD, lo cual validó operativamente que la base de datos es capaz de gestionar usuarios, notas, asistencia o comunicaciones bajo las reglas de negocio establecidas.

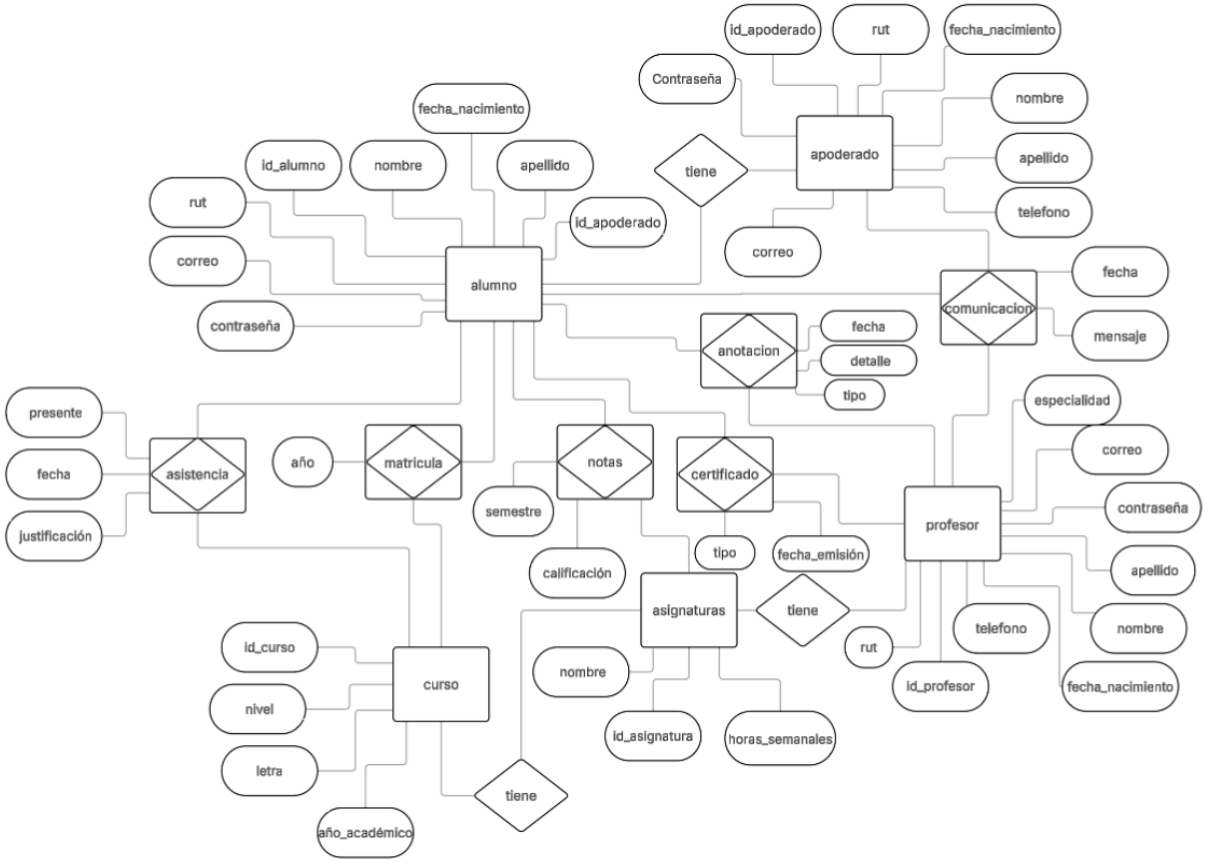
En torno a los resultados obtenidos, es que se cumple con una entrega de una base de datos relacional centralizada, normalizada y operacional, cuya funcionalidad ha sido confirmada a

través de pruebas exitosas de manipulación e integridad de datos, junto a la integración de una interfaz mínima viable desarrollada, valida el modelo físico y muestra la capacidad del sistema para reducir significativamente los errores y agilizar los procesos administrativos. En cuanto al trabajo futuro, si bien la base del sistema es bastante completa y sobre todo funcional, se proyecta la necesidad de avanzar en el desarrollo de una interfaz de usuario completa y robusta que reemplace la versión actual.

En conclusión, el proyecto ha cumplido sus metas de diseño, implementación y validación, proporcionando una solución tecnológica escalable y robusta que sienta las bases para una gestión escolar moderna y eficiente.

Anexos

ANEXO 1 – Diagrama de Entidad-Relación (ER) sobre el Sistema de Gestión de Información Escolar.



ANEXO 2 – Código DDL completo del Sistema de Gestión de Información Escolar.

```
CREATE TABLE apoderado(
        id_apoderado INTEGER PRIMARY KEY AUTOINCREMENT,
        rut VARCHAR(12) UNIQUE NOT NULL,
        nombre VARCHAR(100) NOT NULL,
        apellido VARCHAR(100) NOT NULL,
        correo VARCHAR(255) UNIQUE NOT NULL,
        contraseña TEXT,
        fecha_nacimiento TEXT,
        telefono TEXT
);

CREATE TABLE profesor(
        id_profesor INTEGER PRIMARY KEY AUTOINCREMENT,
        rut VARCHAR(12) UNIQUE NOT NULL,
        nombre VARCHAR(100) NOT NULL,
        apellido VARCHAR(100) NOT NULL,
        correo VARCHAR(225) UNIQUE NOT NULL,
        contraseña TEXT,
        fecha_nacimiento TEXT,
        telefono TEXT
);

CREATE TABLE curso(
        id_curso INTEGER PRIMARY KEY AUTOINCREMENT,
        nombre VARCHAR(100) NOT NULL UNIQUE,
        id_profesor_jefe INTEGER,
        FOREIGN KEY (id_profesor_jefe) REFERENCES profesor(id_profesor)
);

CREATE TABLE alumno(
        id_alumno INTEGER PRIMARY KEY AUTOINCREMENT,
        rut VARCHAR(12) UNIQUE NOT NULL,
        nombre VARCHAR(100) NOT NULL,
        apellido VARCHAR(100) NOT NULL,
        correo VARCHAR(255) UNIQUE NOT NULL,
        fecha_nacimiento DATE NOT NULL,
        id_apoderado INTEGER NOT NULL,
        id_curso INTEGER NOT NULL,
        contraseña TEXT,
        FOREIGN KEY (id_apoderado) REFERENCES apoderado(id_apoderado),
        FOREIGN KEY (id_curso) REFERENCES curso(id_curso)
);
```

```
CREATE TABLE asignatura(
	id_asignatura INTEGER PRIMARY KEY AUTOINCREMENT,
	nombre_asignatura VARCHAR(100) UNIQUE NOT NULL,
	id_profesor_jefe INTEGER,
	FOREIGN KEY (id_profesor_jefe) REFERENCES profesor(id_profesor)
);

CREATE TABLE nota(
	id_nota INTEGER PRIMARY KEY AUTOINCREMENT,
	id_alumno INTEGER NOT NULL,
	id_asignatura INTEGER NOT NULL,
	fecha DATE NOT NULL,
	nota DECIMAL(2,1) NOT NULL CHECK (nota >= 1.0 AND nota <= 7.0),
	FOREIGN KEY (id_alumno) REFERENCES alumno(id_alumno),
	FOREIGN KEY (id_asignatura) REFERENCES asignatura(id_asignatura)
);

CREATE TABLE asistencia(
	id_asistencia INTEGER PRIMARY KEY AUTOINCREMENT,
	id_alumno INTEGER NOT NULL,
	fecha DATE NOT NULL,
	presente BOOLEAN NOT NULL,
	FOREIGN KEY (id_alumno) REFERENCES alumno(id_alumno)
);

CREATE TABLE comunicacion(
	id_comunicacion INTEGER PRIMARY KEY AUTOINCREMENT,
	id_profesor INTEGER NOT NULL,
	id_alumno INTEGER NOT NULL,
	fecha DATE NOT NULL,
	mensaje TEXT NOT NULL,
	FOREIGN KEY (id_profesor) REFERENCES profesor(id_profesor),
	FOREIGN KEY (id_alumno) REFERENCES alumno(id_alumno)
);

CREATE INDEX idx_alumno_rut ON alumno(rut);
CREATE INDEX idx_profesor_rut ON profesor(rut);
CREATE VIEW vista_alumno_detalles AS
SELECT
	alumno.nombre AS nombre_alumno,
	alumno.apellido AS apellido_alumno,
	curso.nombre AS nombre_curso
FROM alumno
JOIN curso ON alumno.id_curso = curso.id_curso;
```

Anexo 3 – Evidencias de carga de datos.

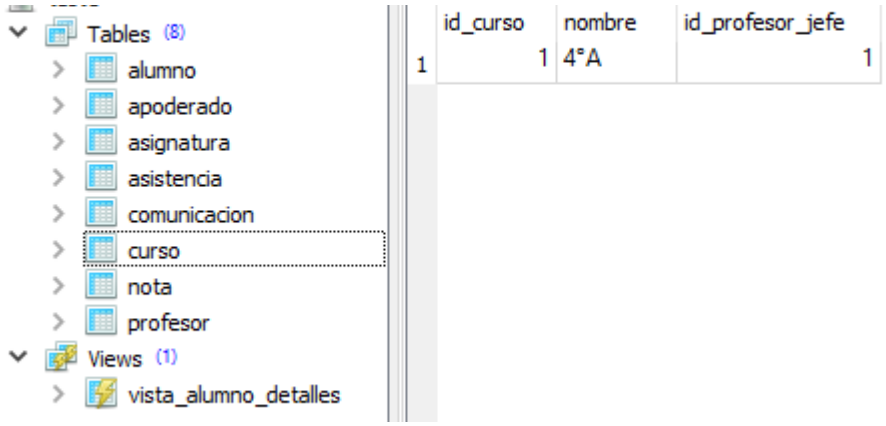


Figura 4: Tabla de curso resultante.

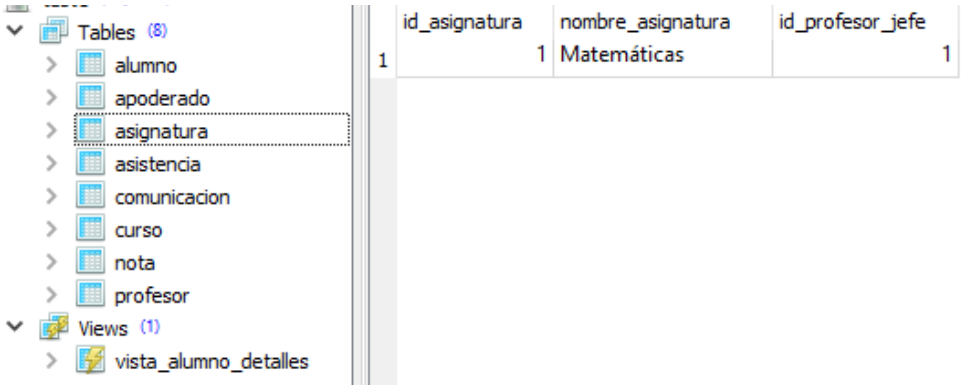


Figura 5: Tabla de asignatura resultante.

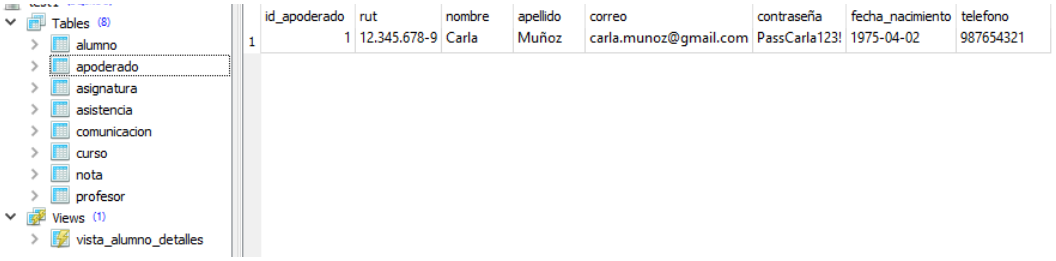


Figura 6: Tabla de apoderado resultante.

Figura 7: Tabla de asistencia resultante.

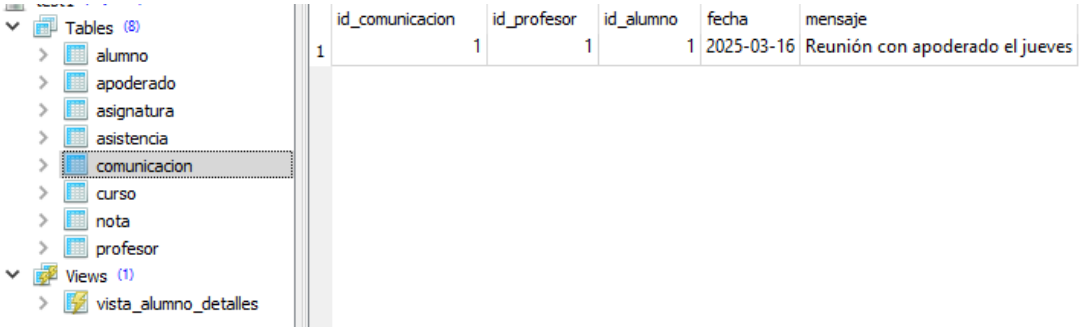


Figura 8: Tabla de comunicación resultante.

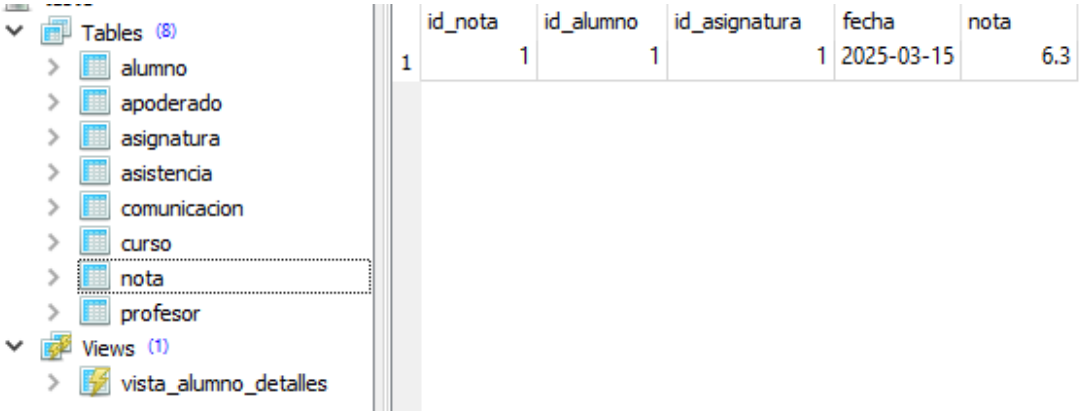


Figura 9: Tabla de nota resultante.

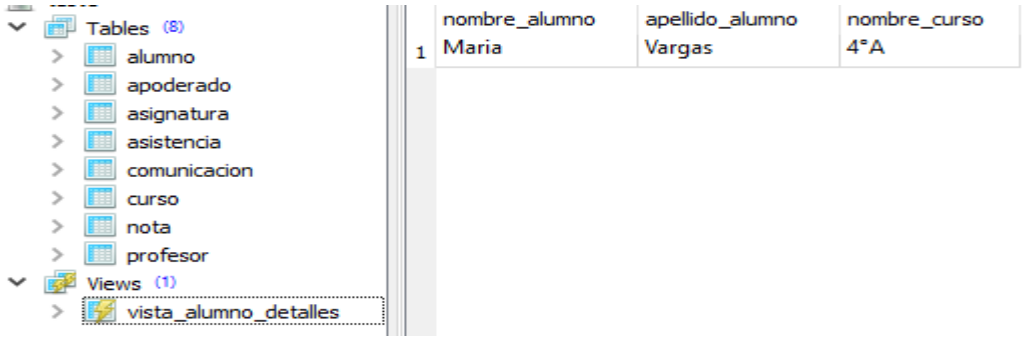


Figura 10: Tabla de vista del alumno resultante.

Anexo 4 - Código SQL del CRUD.

```
-- CREATE: INSERTAR REGISTROS

-- APODERADO
INSERT INTO apoderado (rut, nombre, apellido, correo, contraseña, fecha_nacimiento, telefono)
VALUES ('12.345.678-9', 'Carla', 'Muñoz', 'carla.munoz@gmail.com', 'PassCarla123!', '1975-04-02', '987654321');

-- PROFESOR
INSERT INTO profesor (rut, nombre, apellido, correo, contraseña, fecha_nacimiento, telefono)
VALUES ('11.222.333-4', 'Fernando', 'Lagos', 'fernando.lagos@colegio.cl', 'ProfPass!2025', '1980-08-15', '912345678');

-- CURSO
INSERT INTO curso (nombre, id_profesor_jefe)
VALUES ('4°A', 1);

-- ALUMNO
INSERT INTO alumno (rut, nombre, apellido, correo, fecha_nacimiento, id_apoderado, id_curso, contraseña)
VALUES ('22.333.444-5', 'Maria', 'Vargas', 'maria.vargas@estudiantes.cl', '2012-05-10', 1, 1, 'AlumnoPass2025!');

-- ASIGNATURA
INSERT INTO asignatura (nombre_asignatura, id_profesor_jefe)
VALUES ('Matemáticas', 1);

-- NOTA
INSERT INTO nota (id_alumno, id_asignatura, fecha, nota)
VALUES (1, 1, '2025-03-15', 6.3);

-- ASISTENCIA
INSERT INTO asistencia (id_alumno, fecha, presente)
VALUES (1, '2025-03-15', 1);

-- COMUNICACIÓN
INSERT INTO comunicacion (id_profesor, id_alumno, fecha, mensaje)
VALUES (1, 1, '2025-03-16', 'Reunión con apoderado el jueves');


-- READ: CONSULTAR REGISTROS
```

```
-- Ver alumnos con su curso
SELECT alumno.nombre,
       alumno.apellido,
       curso.nombre AS curso
FROM alumno
JOIN curso ON alumno.id_curso = curso.id_curso;

-- Ver notas de los alumnos
SELECT alumno.nombre,
       asignatura.nombre_asignatura,
       nota.nota
FROM nota
JOIN alumno ON nota.id_alumno = alumno.id_alumno
JOIN asignatura ON nota.id_asignatura = asignatura.id_asignatura;

-- Ver asistencia formateada
SELECT alumno.nombre,
       fecha,
       CASE presente
           WHEN 1 THEN 'Presente'
           WHEN 0 THEN 'Ausente'
       END AS asistencia
FROM asistencia
JOIN alumno ON asistencia.id_alumno = alumno.id_alumno;

-- Ver comunicaciones profesor → alumno
SELECT profesor.nombre AS profesor,
       alumno.nombre AS alumno,
       mensaje,
       fecha
FROM comunicacion
JOIN profesor ON comunicacion.id_profesor = profesor.id_profesor
JOIN alumno ON comunicacion.id_alumno = alumno.id_alumno;


-- UPDATE: MODIFICAR REGISTROS

-- Cambiar correo de un alumno
UPDATE alumno
SET correo = 'maria.actualizada@colegio.cl'
WHERE id_alumno = 1;

-- Actualizar el teléfono de un apoderado
```

```
UPDATE apoderado
SET telefono = '912345678'
WHERE id_apoderado = 1;

-- Modificar una nota existente
UPDATE nota
SET nota = 6.8
WHERE id_nota = 1;


-- DELETE: ELIMINAR REGISTROS


-- Eliminar un registro de asistencia
DELETE FROM asistencia
WHERE id_asistencia = 1;

-- Eliminar una comunicación
DELETE FROM comunicacion
WHERE id_comunicacion = 1;

-- Eliminar una asignatura sin notas asociadas
DELETE FROM asignatura
WHERE id_asignatura = 1;
```

Anexo 5 - Código de la interfaz.

```
#Semana 7: Proyecto de Bases de datos
#Millaray Olivares - Valentin latu - Martina Vargas - Matias Morales

import streamlit as st
```

```
import sqlite3
import pandas as pd
import re
from datetime import datetime

#CONFIGURACIÓN RUTA DE LA BASE#
DB_PATH = "Colegio.db"  #Ajustar el nombre de la base de datos

#CONFIGURACIÓN LOGIN DEL ADMIN#
ADMIN_USER = "admin"
ADMIN_PASS = "12345"

#Título en grande#
st.set_page_config(page_title="Colegio - Sistema", layout="wide")


#CSS para centrar tablas y encabezados y alinear formularios#
st.markdown(
    """
    <style>
    h1, h2, h3 { text-align: center; }
    [data-testid="stDataFrame"] table th { text-align: center !important; }
    [data-testid="stDataFrame"] table td { text-align: center !important; }
    .css-18e3th9 { align-items: center; }
    </style>
    """,
    unsafe_allow_html=True
)

#CONEXIÓN Y EJECUCIONES#
def get_conn():
    conn = sqlite3.connect(DB_PATH, check_same_thread=False)
    conn.execute("PRAGMA foreign_keys = ON;")
    return conn

def run_query_df(query, params=()):
    conn = get_conn()
    try:
        df = pd.read_sql_query(query, conn, params=params)
        return df
    except Exception:
        cur = conn.cursor()
        cur.execute(query, params)
        rows = cur.fetchall()
```

```
        cols = [d[0] for d in cur.description] if cur.description else []
        conn.close()
        if cols:
            return pd.DataFrame(rows, columns=cols)
        return pd.DataFrame()

def run_command(query, params=()):
    conn = get_conn()
    cur = conn.cursor()
    cur.execute(query, params)
    conn.commit()
    conn.close()


#COLUMNA CONTRASEÑA
def ensure_password_columns():
    conn = get_conn()
    cur = conn.cursor()
    tables = ["alumno", "apoderado", "profesor"]
    for t in tables:
        try:
            cur.execute(f"PRAGMA table_info({t});")
        except Exception:
            continue
        cols = [r[1] for r in cur.fetchall()]
        if "contraseña" not in cols and "password" not in cols:
            cur.execute(f"ALTER TABLE {t} ADD COLUMN contraseña TEXT;")
    conn.commit()
    conn.close()

#Ejecutar
try:
    ensure_password_columns()
except Exception as e:
    st.warning(f"Advertencia migración: {e}")

# VALIDACIONES Y FORMATOS#
def validate_rut(rut: str) -> bool:
    return bool(re.match(r"^\d{7,8}-[\dkK]$", str(rut).strip()))

def validate_email(email: str) -> bool:
    return bool(re.match(r"^[^@]+@[^@]+\.[^@]+$", str(email).strip()))

def validate_phone(phone: str) -> bool:
```

```
    return bool(re.match(r"^\d{8,9}$", str(phone).strip()))

def parse_date_ddmmyyyy_to_yyyy_mm_dd(fecha_ddmmyyyy: str):
    if fecha_ddmmyyyy is None:
        return None
    s = str(fecha_ddmmyyyy).strip()
    if s == "":
        return None
    try:
        d = datetime.strptime(s, "%d/%m/%Y")
        return d.strftime("%Y-%m-%d")
    except Exception:
        return None

def format_date_yyyy_mm_dd_to_ddmmyyyy(fecha_yyyy_mm_dd: str):
    if fecha_yyyy_mm_dd is None or str(fecha_yyyy_mm_dd).strip() == "":
        return ""
    try:
        d = datetime.strptime(str(fecha_yyyy_mm_dd).strip(), "%Y-%m-%d")
        return d.strftime("%d/%m/%Y")
    except Exception:
        return ""

#METADATOS TABLAS (campos y requerimientos)#
TABLES = {
    "apoderado": {
        "pk": "id_apoderado",
        "cols": [
            ("rut", "Rut: 11111111-1"),
            ("nombre", "Nombre"),
            ("apellido", "Apellido"),
            ("correo", "Correo: ejemplo@gmail.com"),
            ("telefono", "Teléfono: 8-9 dígitos"),
            ("fecha_nacimiento", "Fecha de nacimiento (DD/MM/YYYY)"),
            ("contraseña", "Contraseña: Con letras, números y símbolos combinados.")
        ],
            "required": ["rut", "nombre", "apellido", "correo", "telefono", "fecha_nacimiento",
"contraseña"]
    },

    "profesor": {
        "pk": "id_profesor",
        "cols": [
            ("rut", "Rut: 11111111-1"),
```

```
            ("nombre", "Nombre"),
            ("apellido", "Apellido"),
            ("correo", "Correo: ejemplo@gmail.com"),
            ("telefono", "Teléfono: 8-9 dígitos"),
            ("fecha_nacimiento", "Fecha de nacimiento (DD/MM/YYYY)"),
            ("contraseña", "Contraseña: Con letras, números y símbolos combinados.")
        ],
            "required": ["rut", "nombre", "apellido", "correo", "telefono", "fecha_nacimiento",
"contraseña"]
    },

    "curso": {
        "pk": "id_curso",
        "cols": [
            ("nombre", "Nombre del curso"),
            ("id_profesor_jefe", "ID profesor jefe (ej: 1)")
        ],
        "required": ["nombre"]
    },

    "alumno": {
        "pk": "id_alumno",
        "cols": [
            ("rut", "Rut: 11111111-1"),
            ("nombre", "Nombre"),
            ("apellido", "Apellido"),
            ("correo", "Correo: ejemplo@gmail.com"),
            ("fecha_nacimiento", "Fecha de nacimiento (DD/MM/YYYY)"),
            ("id_apoderado", "ID apoderado"),
            ("id_curso", "ID curso"),
            ("contraseña", "Contraseña: Con letras, números y símbolos combinados.")
        ],
          "required": ["rut", "nombre", "apellido", "correo", "fecha_nacimiento", "id_apoderado",
"id_curso", "contraseña"]
    },

    "asignatura": {
        "pk": "id_asignatura",
        "cols": [
            ("nombre_asignatura", "Nombre de la asignatura"),
            ("id_profesor_jefe", "ID profesor jefe")
        ],
        "required": ["nombre_asignatura"]
    },
```

```
    "nota": {
        "pk": "id_nota",
        "cols": [
            ("id_alumno", "ID alumno"),
            ("id_asignatura", "ID asignatura"),
            ("fecha", "Fecha (DD/MM/YYYY)"),
            ("nota", "Nota (1.0 a 7.0)")
        ],
        "required": ["id_alumno", "id_asignatura", "fecha", "nota"]
    },

    "asistencia": {
        "pk": "id_asistencia",
        "cols": [
            ("id_alumno", "ID alumno"),
            ("fecha", "Fecha (DD/MM/YYYY)"),
            ("presente", "1=Presente / 0=Ausente")
        ],
        "required": ["id_alumno", "fecha", "presente"]
    },

    "comunicacion": {
        "pk": "id_comunicacion",
        "cols": [
            ("id_profesor", "ID profesor"),
            ("id_alumno", "ID alumno"),
            ("fecha", "Fecha (DD/MM/YYYY)"),
            ("mensaje", "Mensaje")
        ],
        "required": ["id_profesor", "id_alumno", "fecha", "mensaje"]
    }
}



#HELPERS CRUD#
def insert_record(table, values):
    vals = values.copy()
    if table == "alumno" and "fecha_nacimiento" in vals:
                                                        vals["fecha_nacimiento"]        =
parse_date_ddmmyyyy_to_yyyy_mm_dd(vals.get("fecha_nacimiento"))
    if table in ("nota","asistencia","comunicacion") and "fecha" in vals:
        vals["fecha"] = parse_date_ddmmyyyy_to_yyyy_mm_dd(vals.get("fecha"))
```

```
    cols = ", ".join(vals.keys())
    placeholders = ", ".join(["?"]*len(vals))
    sql = f"INSERT INTO {table} ({cols}) VALUES ({placeholders})"
    run_command(sql, tuple(vals.values()))

def get_record_by_pk(table, pk_value):
    pk = TABLES[table]["pk"]
    return run_query_df(f"SELECT * FROM {table} WHERE {pk} = ?", (pk_value,))

def update_record(table, pk_value, values):
    vals = values.copy()
    if "fecha_nacimiento" in vals:
                                                        vals["fecha_nacimiento"]          =
parse_date_ddmmyyyy_to_yyyy_mm_dd(vals.get("fecha_nacimiento"))

    if table in ("nota","asistencia","comunicacion") and "fecha" in vals:
        vals["fecha"] = parse_date_ddmmyyyy_to_yyyy_mm_dd(vals.get("fecha"))
    pk = TABLES[table]["pk"]
    set_clause = ", ".join([f"{k} = ?" for k in vals.keys()])
    params = tuple(vals.values()) + (pk_value,)
    run_command(f"UPDATE {table} SET {set_clause} WHERE {pk} = ?", params)

def delete_record(table, pk_value):
    pk = TABLES[table]["pk"]
    run_command(f"DELETE FROM {table} WHERE {pk} = ?", (pk_value,))

#VALIDACIONES PRE-INSERT/UPDATE#
def validate_inputs(table, data):
    errors = []
    for r in TABLES[table]["required"]:
        if str(data.get(r, "")).strip() == "":
            errors.append(f"Campo requerido vacío: {r}")

    if "rut" in data and data.get("rut") and not validate_rut(data.get("rut")):
        errors.append("Formato RUT inválido (ej: 11111111-1)")

    if "correo" in data and data.get("correo") and not validate_email(data.get("correo")):
        errors.append("Formato correo inválido")

    if "telefono" in data and data.get("telefono") and not validate_phone(data.get("telefono")):
        errors.append("Teléfono inválido (8-9 dígitos)")

    if "fecha_nacimiento" in data and data.get("fecha_nacimiento"):
        if parse_date_ddmmyyyy_to_yyyy_mm_dd(data.get("fecha_nacimiento")) is None:
```

```
            errors.append("Fecha de nacimiento inválida (usar DD/MM/YYYY)")

    if "fecha" in data and data.get("fecha"):
        if parse_date_ddmmyyyy_to_yyyy_mm_dd(data.get("fecha")) is None:
            errors.append("Fecha inválida (usar DD/MM/YYYY)")

    if "nota" in data and data.get("nota"):
        try:
            v = float(data.get("nota"))
            if v < 1.0 or v > 7.0:
                errors.append("La nota debe estar entre 1.0 y 7.0")
        except Exception:
            errors.append("La nota debe ser numérica")

    #Validación asistencia: solo 1 o 0
    if "presente" in data:
        val = str(data.get("presente")).strip()
        if val not in ("1", "0"):
            errors.append("El campo 'Presente' solo acepta 1 = Presente o 0 = Ausente.")

    fk_map = {
        "alumno": [("id_apoderado","apoderado"), ("id_curso","curso")],
        "curso": [("id_profesor_jefe","profesor")],
        "asignatura": [("id_profesor_jefe","profesor")],
        "nota": [("id_alumno","alumno"), ("id_asignatura","asignatura")],
        "asistencia": [("id_alumno","alumno")],
        "comunicacion": [("id_profesor","profesor"), ("id_alumno","alumno")]
    }

    if table in fk_map:
        for col, ref in fk_map[table]:
            val = data.get(col, None)
            if val is not None and str(val).strip() != "":
                df = run_query_df(f'SELECT 1 FROM {ref} WHERE {TABLES[ref]['pk']} = ?
LIMIT 1", (val,))
                if df.empty:
                    errors.append(f"FK inválida: {col}={val} no existe en {ref}")
    return errors

#AUTENTICACIÓN#
def authenticate(username, password):
    u = str(username).strip()
    p = str(password).strip()
    if u == ADMIN_USER and p == ADMIN_PASS:
```

```
        return "admin", {"usuario": ADMIN_USER, "nombre": "Administrador", "apellido": ""}
      df = run_query_df("SELECT * FROM profesor WHERE rut = ? AND contraseña = ? LIMIT 1", (u, p))
    if not df.empty:
        rec = df.iloc[0].to_dict()
        rec.setdefault("nombre", "")
        rec.setdefault("apellido", "")
        return "profesor", rec
     df = run_query_df("SELECT * FROM apoderado WHERE rut = ? AND contraseña = ? LIMIT 1", (u, p))
    if not df.empty:
        rec = df.iloc[0].to_dict()
        rec.setdefault("nombre", "")
        rec.setdefault("apellido", "")
        return "apoderado", rec
      df = run_query_df("SELECT * FROM alumno WHERE rut = ? AND contraseña = ? LIMIT 1", (u, p))
    if not df.empty:
        rec = df.iloc[0].to_dict()
        rec.setdefault("nombre", "")
        rec.setdefault("apellido", "")
        return "alumno", rec
    return None, None

#MOSTRAR DATAFRAME#
def show_dataframe(df, role=None):
    """
    Muestra un dataframe en la UI.
    - Convierte fechas a DD/MM/YYYY
    - Si role == 'alumno' o 'apoderado' convierte 'presente' 1/0 a Presente/Ausente
    - Si role == 'admin' deja los valores crudos (1/0)
    """
    if df is None or df.empty:
        st.info("No hay registros.")
        return
    #Clonar para no modificar original en memoria
    df2 = df.copy()
    for c in df2.columns:
        if "fecha" in c:
            df2[c] = df2[c].apply(format_date_yyyy_mm_dd_to_ddmmyyyy)
    if role in ("alumno", "apoderado"):
        if "presente" in df2.columns:
            # Acepta"0"/"1"
            def map_pres(x):
```

```
            try:
                xi = int(x)
            except:
                xi = 1 if str(x).strip().lower() in ("true","t","yes","y") else 0
            return "Presente" if xi == 1 else "Ausente"
        df2["presente"] = df2["presente"].apply(map_pres)
    st.dataframe(df2, use_container_width=True)


#CARGAR EN BASE AL LOGIN#
if "logged" not in st.session_state:
    st.session_state["logged"] = False
    st.session_state["rol"] = None
    st.session_state["user"] = None

if "action" not in st.session_state:
    st.session_state["action"] = None

if "last_table" not in st.session_state:
    st.session_state["last_table"] = None

if "loaded_row" not in st.session_state:
    st.session_state["loaded_row"] = None

#PANTALLA DE LOGIN#
st.title("Sistema de Gestión - Colegio")
if not st.session_state["logged"]:
    with st.form("login_form"):
        usuario = st.text_input("Usuario (Rut)", placeholder="11111111-1")
          contraseña = st.text_input("Contraseña", type="password", placeholder="*********",
help="Con letras, números y símbolos combinados.")
        submit_login = st.form_submit_button("Ingresar")
    if submit_login:
        rol, user_rec = authenticate(usuario, contraseña)
        if rol:
            st.session_state["logged"] = True
            st.session_state["rol"] = rol
            st.session_state["user"] = user_rec
            st.success("Acceso correcto.")
            st.rerun()
        else:
            st.error("Datos inválidos.")
    st.stop()
```

```
col_a, col_b = st.columns([4,1]) #No hace nada, es por comodidad del boton de cerrar sesión
with col_b:
    if st.button("Cerrar sesión"): #Boton de cerrar sesión
        st.session_state["logged"] = False
        st.session_state["rol"] = None
        st.session_state["user"] = None
        st.session_state["action"] = None
        st.session_state["loaded_row"] = None
        st.rerun()

#VISTA DE ADMIN (CRUD COMPLETO)#
def view_admin():
    st.header("Vista de Administrador")
    st.markdown("Gestionar datos de:")

    table_choice = st.selectbox("Seleccione una tabla", list(TABLES.keys()), index=0)

    if st.session_state["last_table"] != table_choice:
        st.session_state["action"] = None
        st.session_state["loaded_row"] = None
        st.session_state["last_table"] = table_choice

    #Ejemplos y formatos
    with st.expander("Ejemplos y formatos"):
        meta = TABLES[table_choice]

        #Mostrar campos principales por tabla
        for col, hint in meta["cols"]:
            label = col.capitalize()
            if col == "rut":
                st.write("Rut: RUT (ej: 11111111-1)")
            elif col == "nombre":
                st.write("Nombre: Nombre")
            elif col == "apellido":
                st.write("Apellido: Apellido")
            elif col == "correo":
                st.write("Correo: ejemplo@gmail.com")
            elif col == "telefono":
                st.write("Teléfono: 8-9 dígitos")
            elif col == "fecha_nacimiento" or ("fecha" in col and "nacimiento" in col):
                st.write("Fecha de nacimiento: DD/MM/YYYY")
            elif "fecha" in col:
                st.write("Fecha: DD/MM/YYYY")
```

```
            elif col == "contraseña":
                st.write("Contraseña: Con letras, números y símbolos combinados.")
            elif col == "presente":
                st.write("Asistencia: 1 = Presente, 0 = Ausente")
            else:
                #Si es un id o campo genérico, mostrar el hint simplificado
                st.write(f"{label}: {hint}")

        #Mensajes finales según corresponda
        if any("fecha" in c[0] for c in meta["cols"]):
            st.write("\nFechas: DD/MM/YYYY.")
        if any(c[0] == "contraseña" for c in meta["cols"]):
            st.write("\nContraseña: Con letras, números y símbolos combinados.")

#FORMULARIO#
def form_inputs(table, loaded=None):
    meta = TABLES[table]
    inputs = {}
    for col, hint in meta["cols"]:
        label = col.capitalize()
        default = "" if loaded is None else str(loaded.get(col, "") or "")

        if col == "contraseña":
            inputs[col] = st.text_input(
                "Contraseña",
                value=default,
                type="password",
                placeholder="*********",
                help="Con letras, números y símbolos combinados."
            )
        elif "fecha" in col:
            inputs[col] = st.text_input(
                label,
                value=default if "/" in default else "",
                placeholder="DD/MM/YYYY"
            )
        else:
            inputs[col] = st.text_input(label, value=default, placeholder=hint)
    return inputs
#BOTONES#
def crud_section(table):
    st.subheader(f"Tabla: {table.capitalize()}")
```

```
    action = st.radio("Acción", ["Crear", "Leer", "Actualizar", "Eliminar", "Mostrar Todo"], horizontal=True)

    #CREAR
    if action == "Crear":
        st.write("### Crear registro")
        data = form_inputs(table)

        if st.button("Confirmar creación"):
            errors = validate_inputs(table, data)
            if errors:
                for e in errors:
                    st.error(e)
            else:
                insert_record(table, data)
                st.success("Registro creado correctamente.")

    #LEER
    elif action == "Leer":
        st.write("### Leer registro por ID")

        pk = TABLES[table]["pk"]
        pk_value = st.text_input(f"Ingrese {pk}")

        if st.button("Buscar"):
            if pk_value.strip() == "":
                st.warning("Ingrese un ID.")
            else:
                df = get_record_by_pk(table, pk_value)
                show_dataframe(df, role=st.session_state["rol"])

    #ACTUALIZAR
    elif action == "Actualizar":
        st.write("### Actualizar registro")

        pk = TABLES[table]["pk"]
        pk_value = st.text_input(f"Ingrese {pk} a actualizar")

        if st.button("Cargar datos"):
            df = get_record_by_pk(table, pk_value)
            if df.empty:
                st.error("ID no encontrado.")
            else:
                st.session_state["loaded_row"] = df.iloc[0].to_dict()
```

```
        if st.session_state["loaded_row"]:
            st.write("### Datos actuales:")
            show_dataframe(pd.DataFrame([st.session_state["loaded_row"]]))

            st.write("### Modificar:")
            new_data = form_inputs(table, st.session_state["loaded_row"])

            if st.button("Confirmar actualización"):
                errors = validate_inputs(table, new_data)
                if errors:
                    for e in errors:
                        st.error(e)
                else:
                    update_record(table, pk_value, new_data)
                    st.success("Registro actualizado correctamente.")

    #ELIMINAR
    elif action == "Eliminar":
        st.write("### Eliminar registro")

        pk = TABLES[table]["pk"]
        pk_value = st.text_input(f"Ingrese {pk} a eliminar")

        if st.button("Eliminar"):
            delete_record(table, pk_value)
            st.success("Registro eliminado correctamente.")

    #MOSTRAR TODO
    elif action == "Mostrar Todo":
        df = run_query_df(f"SELECT * FROM {table}")
        show_dataframe(df, role=st.session_state["rol"])


#VISTA ADMIN#
def view_admin():
    st.header("Vista de Administrador")
    st.markdown("Gestionar datos de:")

    table_choice = st.selectbox("Seleccione una tabla", list(TABLES.keys()))

    if st.session_state["last_table"] != table_choice:
        st.session_state["action"] = None
        st.session_state["loaded_row"] = None
```

```
        st.session_state["last_table"] = table_choice

    #Ejemplos y formatos
    with st.expander("Ejemplos y formatos"):
        meta = TABLES[table_choice]
        for col, hint in meta["cols"]:
            col_n = col.capitalize()
            if col == "rut":
                st.write("Rut: RUT (ej: 11111111-1)")
            elif col == "correo":
                st.write("Correo: ejemplo@gmail.com")
            elif col == "telefono":
                st.write("Teléfono: 8-9 dígitos")
            elif "fecha" in col:
                st.write("Fecha: DD/MM/YYYY")
            elif col == "contraseña":
                st.write("Contraseña: Con letras, números y símbolos combinados.")
            elif col == "presente":
                st.write("Presente: 1 = Presente / 0 = Ausente")
            else:
                st.write(f"{col_n}: {hint}")

    crud_section(table_choice)

#VISTA PROFESOR#
def view_profesor():
    st.header("Vista de Profesor")

        st.write("Aquí puede ver los cursos, alumnos, asignaturas, notas, asistencia y comunicaciones asignadas.")

    #Profesor puede ver todo menos modificar alumnos/apoderados
    options = ["curso", "asignatura", "nota", "asistencia", "comunicacion"]
    table_choice = st.selectbox("Seleccione tabla a visualizar", options)

    df = run_query_df(f'SELECT * FROM {table_choice}')
    show_dataframe(df, role="profesor")

#VISTA APODERADO#
def view_apoderado():
    usr = st.session_state["user"]
    id_apo = usr["id_apoderado"]

    st.header(f'Vista de Apoderado: {usr['nombre']} {usr['apellido']}')
```

```
    #Obtener sus alumnos
        alumnos = run_query_df("SELECT * FROM alumno WHERE id_apoderado = ?",
(id_apo,))

    if alumnos.empty:
        st.warning("No hay alumnos asociados.")
        return

    st.write("### Sus alumnos:")
    show_dataframe(alumnos, role="apoderado")
    alumno_ids = alumnos["id_alumno"].tolist()

    #Notas
    notas = run_query_df(
        f"SELECT * FROM nota WHERE id_alumno IN ({','.join(['?']*len(alumno_ids))})",
        alumno_ids
    )
    st.write("### Notas:")
    show_dataframe(notas, role="apoderado")

    #Asistencia
    asistencia = run_query_df(
                            f"SELECT * FROM asistencia WHERE id_alumno IN
({','.join(['?']*len(alumno_ids))})",
        alumno_ids
    )
    st.write("### Asistencia:")
    show_dataframe(asistencia, role="apoderado")

    #Comunicaciones
    comunic = run_query_df(
                          f"SELECT * FROM comunicacion WHERE id_alumno IN
({','.join(['?']*len(alumno_ids))})",
        alumno_ids
    )
    st.write("### Comunicaciones:")
    show_dataframe(comunic, role="apoderado")

#VISTA ALUMNO#
def view_alumno():
    usr = st.session_state["user"]
    id_al = usr["id_alumno"]
```

```
    st.header(f"Vista de Alumno: {usr['nombre']} {usr['apellido']}")

    #Datos personales
    datos = run_query_df("SELECT * FROM alumno WHERE id_alumno = ?", (id_al,))
    st.write("### Datos personales")
    show_dataframe(datos, role="alumno")

    #Notas
    notas = run_query_df("SELECT * FROM nota WHERE id_alumno = ?", (id_al,))
    st.write("### Notas")
    show_dataframe(notas, role="alumno")

    #Asistencia
        asistencia = run_query_df("SELECT * FROM asistencia WHERE id_alumno = ?",
(id_al,))
    st.write("### Asistencia")
    show_dataframe(asistencia, role="alumno")

    #Comunicaciones
      comunic = run_query_df("SELECT * FROM comunicacion WHERE id_alumno = ?",
(id_al,))
    st.write("### Comunicaciones")
    show_dataframe(comunic, role="alumno")

#RENDER PRINCIPAL SEGÚN ROL#
rol = st.session_state["rol"]

if rol == "admin":
    view_admin()
elif rol == "profesor":
    view_profesor()
elif rol == "apoderado":
    view_apoderado()
elif rol == "alumno":
    view_alumno()
else:
    st.error("Rol desconocido.")
```

Anexo 6 - Link de GitHub.
https://github.com/MatiasFabianMoralesConcha/Proyecto-de-Base-de-Datos