Universidad Finis Terrae
Curso: Base de datos
Segundo semestre de 2025

Profesor: Fernando San Martin
Ayudante: Juan Eulogio Rivera Vasquez
NRC: 78092

Informe Semana 6 - Proyecto Base de datos

Sistema de Gestión para Gimnasio

Fecha: 23-10-2025
Autores:
- Alan Oliva
- Tomas Tamayo
- Esteban Marques

Emails: ttmayoa@uft.edu , emarques_@uft.edu , aolivah@uft.edu

Índice

1. Introducción

La gestión eficiente de información constituye un factor crítico para la operación exitosa de cualquier organización de servicios. En el sector de gimnasios y centros de acondicionamiento físico, la administración adecuada de datos relacionados con socios, membresías, pagos, clases y reservas resulta fundamental para garantizar tanto la calidad del servicio como la sostenibilidad del negocio. Sin embargo, muchos establecimientos pequeños y medianos continúan operando mediante sistemas manuales o herramientas no integradas que limitan su capacidad de gestión y comprometen la experiencia del usuario.

El presente proyecto aborda esta problemática mediante el análisis de un caso real: un gimnasio orientado a clases grupales que enfrenta dificultades operativas derivadas de la falta de centralización de su información. El establecimiento ofrece diversas modalidades de entrenamiento —funcional, spinning e HIIT— supervisadas por entrenadores especializados, y administra múltiples planes de membresía con beneficios diferenciados. Actualmente, la gestión de socios, reservas, pagos y programación de clases se realiza mediante métodos fragmentados tales como planillas de Excel, agendas físicas y aplicaciones de mensajería, generando inconsistencias en los registros, dificultades en el seguimiento de pagos y control inadecuado de la disponibilidad de cupos.

Los gimnasios modernos presentan diferentes modelos de funcionamiento. Algunos operan bajo un esquema de acceso libre, donde los usuarios emplean el equipamiento de forma autónoma, mientras que otros, como el caso de estudio, se especializan en clases grupales con sesiones programadas y capacidad limitada de participantes. Este segundo modelo requiere una coordinación más compleja entre disponibilidad de entrenadores, gestión de cupos, validación de membresías activas y registro de asistencia, lo que hace indispensable contar con un sistema de información robusto y confiable.

Ante este escenario, el proyecto propone el diseño e implementación de una base de datos relacional normalizada que permita centralizar la gestión operativa del gimnasio. La solución contempla la automatización de procesos críticos, la garantía de integridad referencial de los datos y el desarrollo de una interfaz de usuario intuitiva que facilite las tareas administrativas. A través de este sistema, se busca mejorar la eficiencia organizacional, reducir errores operativos, fortalecer el control financiero y, en última instancia, optimizar la experiencia de los socios y del personal administrativo.

1.1. Objetivos del Proyecto

1.1.1. Objetivo General

Diseñar e implementar una base de datos relacional normalizada hasta la Tercera Forma Normal (3FN) que permita centralizar y automatizar la gestión operativa de un gimnasio orientado a clases grupales, mejorando la eficiencia administrativa, la integridad de los datos y la experiencia de los socios.

1.1.2. Objetivos Específicos

- **Modelado de Datos.** Diseñar un modelo entidad–relación que represente de forma precisa las entidades, atributos y relaciones del gimnasio, garantizando la coherencia lógica del sistema y la correcta correspondencia con los requerimientos funcionales definidos.
- **Normalización y Diseño Relacional.** Transformar el modelo conceptual en un esquema relacional normalizado hasta la Tercera Forma Normal (3FN), definiendo claves primarias, foráneas y restricciones de integridad que aseguren la consistencia y eliminación de redundancias.
- **Implementación de la Base de Datos.** Crear la estructura física en SQLite mediante sentencias DDL, incorporando tablas, índices y relaciones que permitan almacenar y manipular los datos del gimnasio de manera eficiente y segura.
- **Gestión de Información Operativa.** Desarrollar las operaciones esenciales de la base de datos para la administración de socios, planes, suscripciones, pagos, clases y reservas, cumpliendo con las reglas de negocio definidas.
- **Control Automatizado de Reservas.** Implementar validaciones automáticas que verifiquen el estado de la suscripción, el límite de clases mensuales y la disponibilidad de cupos antes de confirmar una reserva, asegurando la integridad del proceso y evitando sobrecupos o inconsistencias.
- **Optimización del Sistema.** Aplicar mecanismos de integridad referencial, restricciones y consultas optimizadas que garanticen un acceso eficiente y confiable a la información, reduciendo errores y tiempos de respuesta.

2. Definición del Problema

2.1. Descripción del problema

El gimnasio objeto de estudio se caracteriza por ofrecer una amplia variedad de servicios, entre los que se incluyen clases grupales de tipo funcional, spinning e HIIT, además de la supervisión de entrenadores especializados y la administración de diversos planes de membresía. Esta diversidad de actividades implica el manejo de un volumen considerable de información y una estructura de gestión compleja que, en la actualidad, no se encuentra debidamente controlada.

La administración de los datos relacionados con socios, suscripciones, pagos, reservas y planificación de clases se lleva a cabo mediante procedimientos manuales y herramientas no integradas, tales como planillas de Excel, agendas físicas y aplicaciones de mensajería. La utilización de estos medios fragmentados dificulta la actualización y el acceso a la información, afectando la consistencia y fiabilidad de los registros.

Esta falta de integración provoca múltiples inconvenientes operativos, entre los que destacan la duplicación o pérdida de datos, la dificultad para realizar seguimientos de pagos y la ineficiencia en la gestión de reservas. Asimismo, la disponibilidad de cupos en las clases grupales no se controla de manera precisa, lo que ocasiona tanto sobrecupo como baja ocupación en las sesiones programadas.

El escenario descrito produce una doble insatisfacción: por un lado, de los socios, quienes experimentan una gestión desorganizada del servicio; y por otro, del personal administrativo, que debe operar con sistemas ineficaces y sin trazabilidad. Ante esta problemática, se identifica la necesidad de implementar un sistema unificado sustentado en una base de datos relacional, capaz de gestionar de forma automatizada, coherente y segura todos los procesos vinculados a socios, membresías, pagos, reservas, entrenadores y clases.

2.2. Necesidades y Desafíos Detectados

A partir del análisis del sistema actual de gestión del gimnasio, se identificaron diversas dificultades que afectan la eficiencia operativa y la integridad de la información.

Falta de centralización de datos: La información se encuentra dispersa en diferentes medios, como planillas de Excel, agendas físicas y aplicaciones de mensajería. Esta fragmentación dificulta la actualización, el acceso y el control de los registros, generando inconsistencias cuando distintos miembros del personal modifican la información de manera simultánea sin mecanismos de validación o sincronización.

Errores frecuentes en reservas y pagos: La ausencia de controles automáticos provoca sobrecupos, omisiones en los pagos y confusiones respecto a la vigencia de las membresías. Al no existir un sistema centralizado, se posibilita la confirmación de reservas para socios con membresías vencidas o con cupos agotados, lo que deriva en conflictos administrativos y en un deterioro de la experiencia del usuario.

Limitada trazabilidad y seguridad de los datos: No se dispone de mecanismos de respaldo automatizado ni de una gestión adecuada de los permisos de acceso a información sensible de los socios. Esta situación compromete tanto la operatividad como la privacidad de los datos. Asimismo, la carencia de registros históricos confiables impide realizar auditorías, identificar tendencias y recuperar información ante eventuales pérdidas o errores.

3. Requisitos del Sistema

3.1. Requisitos Funcionales

RF-01: Gestión de Socios
El sistema deberá registrar socios con información como RUT (identificador único), nombre, datos de contacto y fecha de nacimiento. También deberá permitir modificar datos existentes, consultar historial de membresías, pagos y actividades, y dar de baja a socios manteniendo su historial para futura referencia.

RF-02: Gestión de Planes de Membresía
El sistema deberá administrar distintos planes de membresía, diferenciados por precio, duración y beneficios. Cada socio podrá tener una única membresía activa simultáneamente. El sistema deberá registrar las fechas de inicio y fin, y mostrar el estado de la membresía (activa o vencida).

RF-03: Pagos Asociados a Membresía
El sistema deberá registrar pagos efectuados por socios, indicando fecha, monto, medio de pago y estado. Cada pago estará vinculado a la activación o renovación de la membresía, asegurando trazabilidad del historial financiero. El sistema deberá permitir renovar una membresía al registrar un nuevo pago, actualizando automáticamente las fechas de vigencia y manteniendo el historial de renovaciones.

RF-04: Gestión de Entrenadores
El sistema deberá registrar entrenadores con su perfil profesional y especialidad, permitiendo asignarlos a clases en función de su disponibilidad y consultar el listado de clases en las que participan.

RF-05: Gestión de Clases Grupales
El sistema deberá permitir programar clases grupales definiendo tipo de actividad (funcional, spinning, HIIT), horario, duración, cupo máximo y entrenador asignado. Deberá ser posible crear, modificar o cancelar clases según sea necesario, y consultar la disponibilidad de cupos en tiempo real.

RF-06: Sistema de Reservas y Asistencia
El sistema deberá permitir a los socios reservar clases grupales, validando automáticamente que la membresía esté activa, no se haya superado el límite de clases mensuales según el plan contratado, y existan cupos disponibles en la clase seleccionada. Además, deberá permitir cancelar reservas con anticipación definida y registrar la asistencia efectiva de los socios a cada clase, vinculando esta información con las reservas realizadas.

RF-07: Consultas y Reportes
El sistema deberá generar consultas y reportes analíticos para apoyar la toma de decisiones, incluyendo reportes de asistencia por clase y período, ingresos mensuales discriminados por tipo de membresía, estadísticas sobre clases más populares y horarios de mayor demanda, listado de socios con pagos pendientes o en estado de morosidad, y proyección de renovaciones de membresía para el mes en curso.

3.2. Requisitos No Funcionales

RNF-01: Usabilidad
El sistema deberá disponer de una interfaz clara, ordenada e intuitiva que facilite la ejecución de las operaciones principales (registro de socios, creación de reservas y consulta de pagos) por parte del personal administrativo sin requerir capacitación extensa. Al menos el 80 % de las tareas frecuentes deberán completarse en tres clics o menos desde el menú principal. Los mensajes de error deberán ser comprensibles y orientarán al usuario respecto a la acción necesaria para corregir el problema.

RNF-02: Rendimiento
El sistema deberá responder a consultas básicas —como la verificación de membresías activas, la disponibilidad de clases o la búsqueda de socios— en un tiempo inferior a tres segundos en un equipo con especificaciones estándar. Los reportes simples no deberán superar los cinco segundos de generación con un volumen de hasta 200 socios y 50 clases mensuales. El diseño deberá contemplar la escalabilidad necesaria para mantener el rendimiento ante un aumento en la cantidad de datos.

RNF-03: Integridad de Datos
Deberán implementarse restricciones de integridad referencial mediante el uso de claves primarias y foráneas. El sistema evitará inconsistencias como pagos sin socio, reservas asociadas a clases inexistentes o membresías asignadas a usuarios no registrados. Se validarán campos obligatorios y formatos esenciales, tales como la estructura correcta del RUT.

RNF-04: Confiabilidad
Las operaciones críticas, como los pagos y las reservas, deberán ejecutarse dentro de transacciones SQL (`BEGIN`, `COMMIT`, `ROLLBACK`) para garantizar que, ante un error, los cambios no se almacenen de forma parcial. El sistema deberá contemplar la creación de copias de seguridad manuales del archivo de base de datos SQLite, las cuales serán responsabilidad del administrador, realizándose al menos una vez por semana.

RNF-05: Mantenibilidad
El código SQL deberá incluir comentarios explicativos sobre la estructura de tablas, vistas y consultas complejas. Se utilizarán convenciones de nombres consistentes (por ejemplo: `tabla_socios`, `id_socio`, `fk_socio_membresia`). El diseño de la base de datos se mantendrá normalizado hasta la Tercera Forma Normal (3FN), y las decisiones de diseño deberán estar documentadas en el informe final.

RNF-06: Portabilidad
El sistema deberá implementarse utilizando SQLite, asegurando su ejecución en entornos Windows, macOS o Linux sin requerir configuraciones complejas más allá de Python 3.11 y las bibliotecas especificadas (Streamlit, Taipy o FastHTML). El archivo `.db` deberá ser autocontenido y fácilmente transferible entre los integrantes del equipo.

RNF-07: Documentación del Proyecto
El proyecto deberá incluir documentación técnica completa, que incorpore el modelo entidad–relación, el esquema relacional normalizado y las principales consultas SQL. Además, se elaborará un manual de usuario básico con capturas de pantalla. La documentación final se entregará en formato PDF y deberá cumplir con las normas de presentación IEEE o APA, incluyendo índice, numeración de figuras y referencias bibliográficas.

RNF-08: Protección de Datos
El sistema no deberá exponer información sensible de los socios (número telefónico completo, correo electrónico o historial de pagos detallado) en listados generales ni en reportes públicos. Los datos completos estarán disponibles únicamente en la vista individual de cada socio. Asimismo, se evitará registrar información sensible en archivos de log o exportaciones automáticas.

4. Modelado de la Base de Datos

4.1. Modelo Entidad–Relación

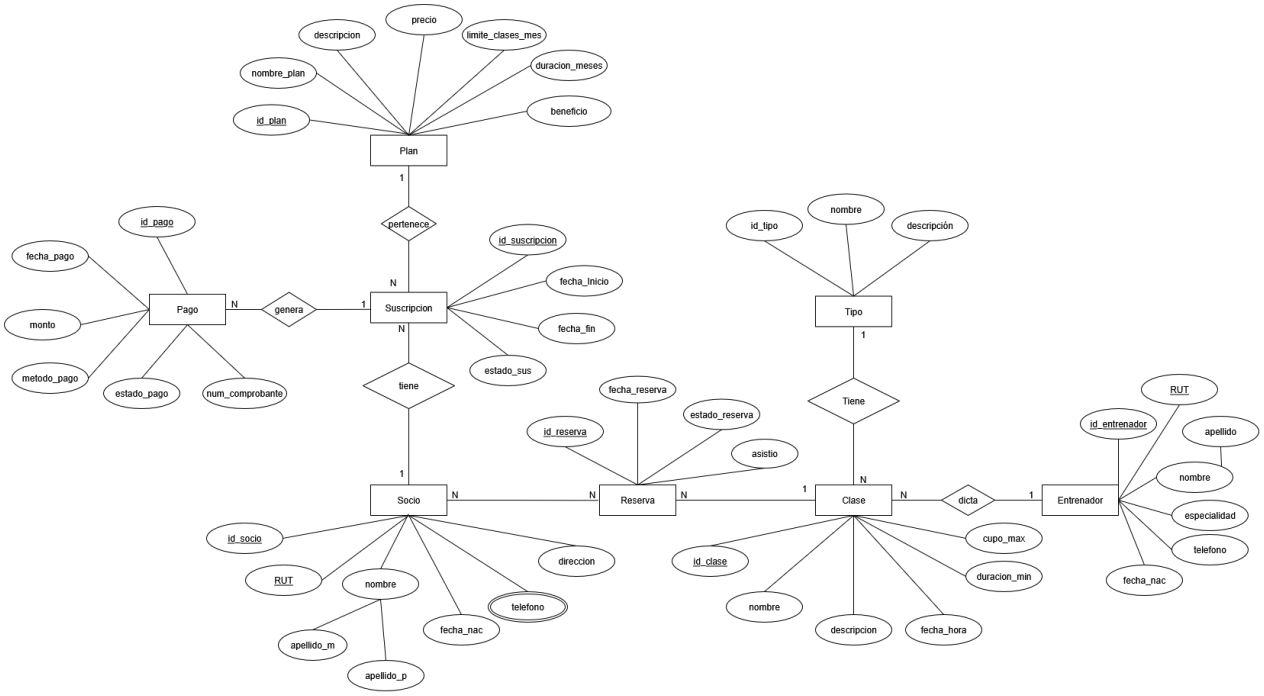

Figura 1: Modelo entidad–relación del sistema de gestión del gimnasio.

El diagrama ER de la Figura 1 describe el dominio mediante ocho entidades: *Socio*, *Plan*, *Suscripción*, *Pago*, *Clase*, *Tipo*, *Entrenador* y *Reserva*. Un *Socio* puede mantener varias *Suscripciones* (1–N); cada *Suscripción* pertenece a un *Plan* (N–1) y genera *Pagos* (1–N). Las *Clases* se clasifican por *Tipo* (1–N) y son dictadas por un *Entrenador* (1–N). La *Reserva* vincula *Socio* con *Clase* y registra estado y asistencia, lo que permite controlar capacidad (`cupo_max`) y trazabilidad operativa.

4.2. Modelo Relacional

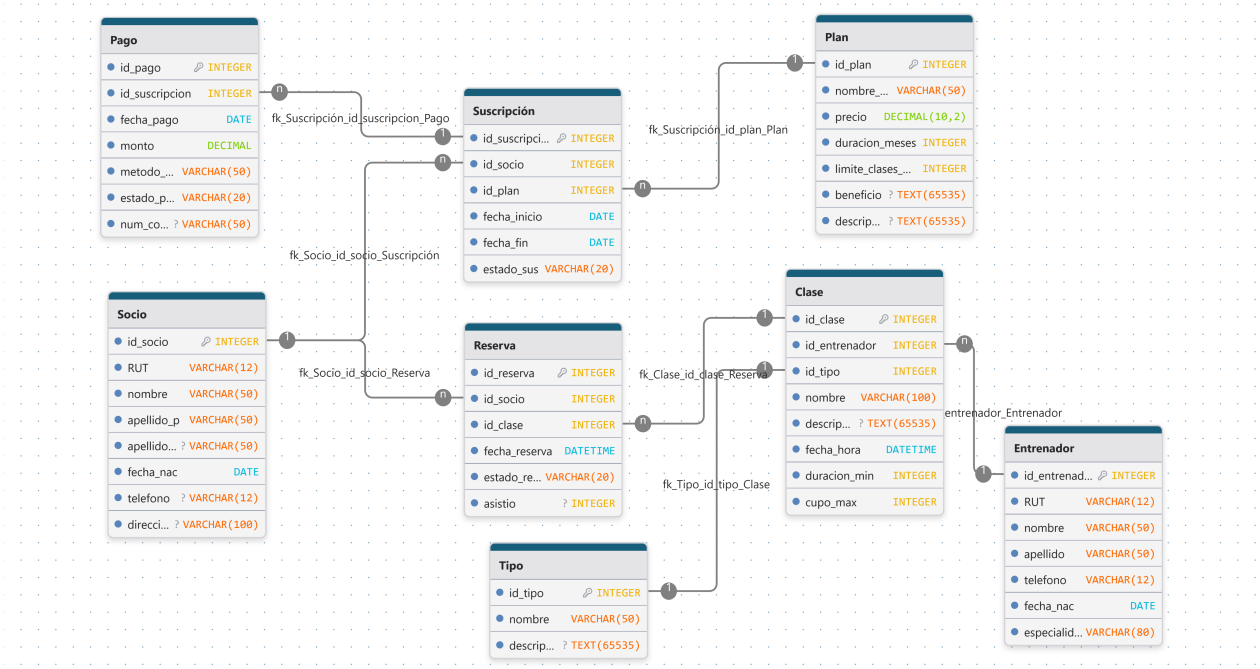

Figura 2: Modelo relacional del sistema de gestión del gimnasio.

El esquema relacional materializa las relaciones del ER con claves primarias numéricas y claves foráneas explícitas: `suscripcion(id_socio)` y `suscripcion(id_plan)`, `pago(id_suscripcion)`, `clase(id_entrenador)`, `clase(id_tipo)`, `reserva(id_socio)` y `reserva(id_clase)`. Se definen dominios y restricciones de integridad: estados como *ENUM* (suscripción/pago/reserva), `precio` como *DECIMAL(10,2)*, control de capacidad mediante `cupo_max`, unicidad en `rut` y `num_comprobante`, y la prevención de doble reserva (índice `UNIQUE(id_socio, id_clase)`). Este diseño es consistente con el ER y está normalizado hasta 3FN, lo que reduce redundancias y habilita consultas de gestión (ingresos por plan, cupos y asistencia).

5. Normalización y Claves

5.1. Normalización del Modelo Físico

El modelo físico del sistema se diseñó aplicando de manera sistemática las tres primeras formas normales (1FN, 2FN y 3FN). Cada tabla fue analizada con el propósito de eliminar redundancias, garantizar dependencias funcionales completas y evitar dependencias transitivas entre atributos no clave. A continuación, se presenta la justificación resumida de la normalización por entidad:

- **Socio**: 1FN por atributos atómicos (`nombre`, `apellido_p`, `direccion`, etc.); 2FN trivial al tener PK simple (`id_socio`); 3FN al no existir dependencias entre atributos no clave. Se garantiza unicidad de `RUT` con una restricción `UNIQUE`.
- **Plan**: 1FN por atributos indivisibles (`precio`, `duracion_meses`, `limite_clases`); 2FN por PK simple (`id_plan`); 3FN al no almacenar datos derivados ni dependencias entre no claves (p. ej., `precio` no depende de `duracion_meses`). El campo `beneficio` se mantiene como texto, permitiendo futuras catalogaciones sin afectar la 3FN.
- **Suscripcion**: 1FN por valores escalares; 2FN al depender todos los atributos de `id_suscripcion`; 3FN al no haber dependencias transitivas (`estado_sus` es independiente de las fechas). Los datos del socio y del plan no se duplican: se referencian mediante FK.
- **Pago**: 1FN por datos financieros atómicos (`monto`, `fecha_pago`); 2FN por PK simple (`id_pago`); 3FN al no existir dependencia entre `metodo_pago` y `estado_pago`. `num_comprobante` es `UNIQUE` para trazabilidad.
- **Clase**: 1FN por campos indivisibles (`fecha_hora`, `duracion_min`, `cupo_max`); 2FN por PK simple (`id_clase`); 3FN ya que `id_tipo` e `id_entrenador` sólo son FKs y no determinan otros atributos almacenados en la misma tabla.
- **Tipo**: 1FN y 2FN por estructura simple (PK `id_tipo`); 3FN al no haber atributos derivados ni transitivos.
- **Entrenador**: 1FN por datos atómicos; 2FN por PK simple; 3FN al no existir dependencias entre atributos no clave. `RUT` se controla con `UNIQUE`.
- **Reserva**: 1FN por atributos atómicos (`fecha_reserva`, `estado_reserva`, `asistio`); 2FN por PK simple (`id_reserva`); 3FN al no haber relación funcional entre `asistio` y `estado_reserva`. Para evitar duplicidades operativas se define `UNIQUE(id_socio, id_clase)`.

En síntesis, todas las tablas cumplen la **Tercera Forma Normal (3FN)**: no hay grupos repetitivos (1FN), no existen dependencias parciales sobre claves compuestas (2FN, trivial por PK simples) ni dependencias transitivas entre atributos no clave (3FN). El uso de FKs preserva la integridad referencial, y las restricciones `UNIQUE` y `CHECK` refuerzan la calidad de los datos conforme a las reglas de negocio.

5.2. Definición de Claves, Restricciones de Integridad y Optimizaciones

Cuadro 1: Claves primarias y foráneas del modelo físico

Entidad	**Clave Primaria**	**Clave(s) Foránea(s)**
Socio	id_socio	–
Entrenador	id_entrenador	–
Plan	id_plan	–
Tipo	id_tipo	–
Suscripcion	id_suscripcion	id_socio (Socio), id_plan (Plan)
Pago	id_pago	id_suscripcion (Suscripcion)
Clase	id_clase	id_tipo (Tipo), id_entrenador (Entrenador)
Reserva	id_reserva	id_socio (Socio), id_clase (Clase)

En el sistema se emplean **claves primarias enteras autoincrementales** (`id_*`) en todas las tablas para garantizar la identificación única y la eficiencia en las relaciones. Los `RUT` de Socio y Entrenador se controlan con `UNIQUE` como identificadores del mundo real, pero no se utilizan como PK.

La **integridad referencial** se asegura mediante claves foráneas que materializan las relaciones 1–N del modelo entidad–relación: una Suscripción referencia a su Socio y Plan; cada Pago referencia a su Suscripción; cada Clase referencia a su Tipo y Entrenador; y cada Reserva referencia a un Socio y a una Clase. De este modo, se evitan registros huérfanos e inconsistencias.

Para la **integridad de dominio**, se definen las siguientes restricciones:

(I) Valores positivos en montos y cupos: `CHECK(monto >0)`, `CHECK(cupo_max >0)`.

(II) Fechas no nulas en eventos: `fecha_hora`, `fecha_pago`, `fecha_reserva`.

(III) Estados controlados mediante `CHECK`: `estado_sus` $\in$ {activa, vencida, cancelada}, `estado_pago` $\in$ {pendiente, completado, rechazado}, `estado_reserva` $\in$ {pendiente, confirmada, cancelada}; y `asistio` como valor booleano (`DEFAULT 0`). Además, `RUT` y `num_comprobante` se declaran `UNIQUE` para asegurar trazabilidad.

En cuanto a las **optimizaciones**, se indexan las FKs de uso frecuente: `suscripcion.id_socio`, `suscripcion.id_plan`, `pago.id_suscripcion`, `clase.id_entrenador`, `clase.id_tipo`, `reserva.id_socio` `reserva.id_clase`. También se recomienda `UNIQUE(id_socio, id_clase)` en Reserva para impedir una doble inscripción del mismo socio en la misma clase. La regla de "una sola suscripción activa por socio" puede implementarse mediante lógica de aplicación o con un trigger. El esquema se encuentra normalizado hasta la **3FN**, reduciendo redundancias y facilitando la eficiencia en las consultas y el mantenimiento.

6. Implementación del Modelo Físico

El modelo físico del sistema fue implementado en **SQLite**, gestionado a través de scripts en Python para asegurar la portabilidad y la carga dinámica de datos. Cada entidad identificada en el modelo conceptual se transformó en una tabla, definiendo sus tipos de datos, claves primarias y foráneas, así como restricciones de integridad y lógica de negocio avanzada mediante Triggers.

A continuación, se presenta un fragmento representativo del código SQL utilizado.

Entidad Socio. La entidad `Socio` representa a cada miembro del gimnasio. Cada socio posee un identificador único (`id_socio`). Para garantizar la integridad financiera y la trazabilidad histórica (RNF-03), se implementó un mecanismo de **Borrado Lógico (Soft Delete)**. Se añadió el atributo `activo` (con valor por defecto 1). Al eliminar un socio, el sistema no destruye el registro, sino que cambia este estado a 0. Esto evita que la eliminación accidental de un socio borre sus pagos históricos.

Algoritmo 1: Tabla Socio (Con Soft Delete)

```
CREATE TABLE Socio (
id_socio   INTEGER       PRIMARY KEY AUTOINCREMENT,
RUT        VARCHAR(12)   NOT NULL UNIQUE,
nombre     VARCHAR(50)   NOT NULL,
apellido_p VARCHAR(50)   NOT NULL,
apellido_m VARCHAR(50),
fecha_nac  DATE          NOT NULL,
telefono   VARCHAR(12),
direccion  VARCHAR(100),
activo     INTEGER       NOT NULL DEFAULT 1, -- Soft Delete
CHECK (LENGTH(RUT) >= 9)
);
```

	Name	Data type	Clave Primaria	Foreign Key	Único	Check	No es NULL	Collate	Generated	
1	id_socio	INTEGER								NULL
2	RUT	VARCHAR (12)								NULL
3	nombre	VARCHAR (50)								NULL
4	apellido_p	VARCHAR (50)								NULL
5	apellido_m	VARCHAR (50)								NULL
6	fecha_nac	DATE								NULL
7	telefono	VARCHAR (12)								NULL
8	direccion	VARCHAR ...								NULL
9	activo	INTEGER								1

Figura 3: Implementación de la tabla `Socio` con atributo de persistencia.

Entidad Suscripción. La entidad `Suscripcion` almacena la información de las membresías. Las restricciones de integridad, como `CHECK (fecha_fin >fecha_inicio)`, aseguran la validez temporal. Aunque se define `ON DELETE CASCADE` para integridad referencial estricta, la aplicación gestiona la baja mediante el borrado lógico del socio, manteniendo los datos históricos accesibles.

Algoritmo 2: Tabla Suscripcion

```
CREATE TABLE Suscripcion (
id_suscripcion INTEGER  PRIMARY KEY AUTOINCREMENT,
id_socio       INTEGER  NOT NULL,
id_plan        INTEGER  NOT NULL,
fecha_inicio   DATE     NOT NULL,
fecha_fin      DATE     NOT NULL,
estado_sus     VARCHAR(20) NOT NULL DEFAULT 'activa',
FOREIGN KEY (id_socio) REFERENCES Socio(id_socio)
ON DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY (id_plan) REFERENCES Plan(id_plan)
ON DELETE RESTRICT ON UPDATE CASCADE,
CHECK (estado_sus IN ('activa','vencida','cancelada')),
CHECK (fecha_fin > fecha_inicio)
);
```

	Name	Data type	Clave Primaria	Foreign Key	Único	Check	No es NULL	Collate	Generated	
1	id_suscripcion	INTEGER								NULL
2	id_socio	INTEGER								NULL
3	id_plan	INTEGER								NULL
4	fecha_inicio	DATE								NULL
5	fecha_fin	DATE								NULL
6	estado_sus	VARCHAR (20)								activa

Figura 4: Implementación de la tabla `Suscripcion` en el modelo físico.

Entidad Clase: La entidad Clase registra la información de cada sesión programada. Mediante las claves foráneas `id_entrenador` e `id_tipo`, cada clase se asocia con un entrenador y un tipo de entrenamiento. Las restricciones CHECK aseguran valores válidos en la duración y el cupo.

Algoritmo 3: Tabla Clase

```
CREATE TABLE Clase (
id_clase      INTEGER       PRIMARY KEY AUTOINCREMENT,
id_entrenador INTEGER       NOT NULL,
id_tipo       INTEGER       NOT NULL,
nombre        VARCHAR(100)  NOT NULL,
descripcion   TEXT,
fecha_hora    DATETIME      NOT NULL,
duracion_min  INTEGER       NOT NULL,
cupo_max      INTEGER       NOT NULL,
FOREIGN KEY (id_entrenador) REFERENCES Entrenador(id_entrenador),
FOREIGN KEY (id_tipo) REFERENCES Tipo(id_tipo),
CHECK (duracion_min > 0),
CHECK (cupo_max > 0)
);
```

	Name	Data type	Clave Primaria	Foreign Key	Único	Check	No es NULL	Collate	Generated	
1	id_clase	INTEGER								NULL
2	id_entrenador	INTEGER								NULL
3	id_tipo	INTEGER								NULL
4	nombre	VARCHAR (100)								NULL
5	descripcion	TEXT								NULL
6	fecha_hora	DATETIME								NULL
7	duracion_min	INTEGER								NULL
8	cupo_max	INTEGER								NULL

Figura 5: Implementación de la tabla Clase.

Entidad Plan. La entidad `Plan` almacena la información de los tipos de membresías. Las restricciones `CHECK` garantizan que los valores numéricos (precio, duración) sean válidos y positivos.

Algoritmo 4: Tabla Plan

```
CREATE TABLE Plan (
id_plan         INTEGER       PRIMARY KEY AUTOINCREMENT,
nombre_plan     VARCHAR(50)   NOT NULL UNIQUE,
precio          DECIMAL(10,2) NOT NULL,
duracion_meses  INTEGER       NOT NULL,
limite_clases   INTEGER,
beneficio       TEXT,
descripcion     TEXT,
CHECK (precio > 0),
CHECK (duracion_meses > 0)
);
```

	Name	Data type	Clave Primaria	Foreign Key	Único	Check	No es NULL	Collate	Generated	
1	id_plan	INTEGER								NULL
2	nombre_plan	VARCHAR (50)								NULL
3	precio	DECIMAL (10,...								NULL
4	duracion_meses	INTEGER								NULL
5	limite_clases	INTEGER								NULL
6	beneficio	TEXT								NULL
7	descripcion	TEXT								NULL

Figura 6: Implementación de la tabla `Plan` en el modelo físico.

Entidad Reserva y Triggers de Control. La entidad `Reserva` vincula a cada socio con una clase. Para robustecer la integridad de los datos más allá de la interfaz gráfica, se implementaron **Triggers (Disparadores)** a nivel de base de datos que validan reglas de negocio críticas antes de cualquier inserción:

1. **Control de Aforo (`trg_check_cupo_insert`):** Previene la sobreventa de cupos en entornos concurrentes, bloqueando la transacción si el cupo máximo ha sido alcanzado.
2. **Validación de Vigencia (`trg_check_vigencia_socio`):** Impide mediante un error de base de datos que socios con membresías vencidas realicen reservas.

Algoritmo 5: Tabla Reserva y Triggers

```
-- Tabla Base
CREATE TABLE Reserva (
id_reserva     INTEGER      PRIMARY KEY AUTOINCREMENT,
id_socio       INTEGER      NOT NULL,
id_clase       INTEGER      NOT NULL,
fecha_reserva  DATETIME     NOT NULL DEFAULT (DATETIME('now')),
estado_reserva VARCHAR(20)  NOT NULL DEFAULT 'confirmada',
asistio        INTEGER      DEFAULT 0,
FOREIGN KEY (id_socio) REFERENCES Socio(id_socio) ON DELETE CASCADE,
FOREIGN KEY (id_clase) REFERENCES Clase(id_clase) ON DELETE CASCADE,
UNIQUE (id_socio, id_clase)
);

-- Trigger Anti-Sobrecupo
CREATE TRIGGER trg_check_cupo_insert BEFORE INSERT ON Reserva
BEGIN
SELECT CASE WHEN (SELECT COUNT(*) FROM Reserva
WHERE id_clase = NEW.id_clase AND estado_reserva = 'confirmada')
>= (SELECT cupo_max FROM Clase WHERE id_clase = NEW.id_clase)
THEN RAISE(ABORT, 'Error: CUPOS_AGOTADOS') END;
END;
```

	Name	Data type	Clave Primaria	Foreign Key	Único	Check	No es NULL	Collate	Generated	
1	id_reserva	INTEGER								NULL
2	id_socio	INTEGER								NULL
3	id_clase	INTEGER								NULL
4	fecha_reserva	DATETIME								DATETIME('now')
5	estado_reserva	VARCHAR (20)								confirmada
6	asistio	INTEGER								0

Figura 7: Implementación de la tabla `Reserva` y lógica de triggers.

Entidad Pago. La entidad `Pago` registra las transacciones financieras. Para asegurar la calidad de los datos contables, se definieron restricciones de dominio estrictas mediante `CHECK`, impidiendo el registro de montos negativos o métodos de pago no válidos.

Algoritmo 6: Tabla Pago

```
CREATE TABLE Pago (
id_pago           INTEGER      PRIMARY KEY AUTOINCREMENT,
id_suscripcion    INTEGER      NOT NULL,
fecha_pago        DATE         NOT NULL DEFAULT (DATE('now')),
monto             DECIMAL(10,2) NOT NULL,
metodo_pago       VARCHAR(50)  NOT NULL,
estado_pago       VARCHAR(20)  NOT NULL DEFAULT 'completado',
num_comprobante   VARCHAR(50),
FOREIGN KEY (id_suscripcion) REFERENCES Suscripcion(id_suscripcion)
ON DELETE CASCADE ON UPDATE CASCADE,
CHECK (monto > 0),
CHECK (metodo_pago IN ('transferencia','webpay','tarjeta','efectivo'))
);
```

	Name	Data type	Clave Primaria	Foreign Key	Único	Check	No es NULL	Collate	Generated	
1	id_pago	INTEGER								NULL
2	id_suscripcion	INTEGER								NULL
3	fecha_pago	DATE								DATE('now')
4	monto	DECIMAL (10,...								NULL
5	metodo_pago	VARCHAR (50)								NULL
6	estado_pago	VARCHAR (20)								completado
7	num_comprobante	VARCHAR (50)								NULL

Figura 8: Implementación de la tabla `Pago` en el modelo físico.

Entidad Tipo y Entrenador. Estas tablas maestras (Catálogos) permiten la normalización de los datos de clases, evitando redundancias.

Algoritmo 7: Tablas Tipo y Entrenador

```
CREATE TABLE Tipo (
id_tipo INTEGER PRIMARY KEY AUTOINCREMENT,
nombre  VARCHAR(50) NOT NULL UNIQUE,
descripcion TEXT
);

CREATE TABLE Entrenador (
id_entrenador INTEGER PRIMARY KEY AUTOINCREMENT,
RUT           VARCHAR(12) NOT NULL UNIQUE,
nombre        VARCHAR(50) NOT NULL,
apellido      VARCHAR(50) NOT NULL,
fecha_nac     DATE        NOT NULL,
especialidad  VARCHAR(80) NOT NULL
);
```

	Name	Data type	Clave Primaria	Foreign Key	Único	Check	No es NULL	Collate	Generated	
1	id_entrenador	INTEGER								NULL
2	RUT	VARCHAR (12)								NULL
3	nombre	VARCHAR (50)								NULL
4	apellido	VARCHAR (50)								NULL
5	telefono	VARCHAR (12)								NULL
6	fecha_nac	DATE								NULL
7	especialidad	VARCHAR (80)								NULL

Figura 9: Implementación de tablas catálogo.

6.1. Vistas del Sistema

Se implementaron vistas para facilitar la generación de reportes y validar la integridad de los datos sin requerir consultas complejas recurrentes.

Historial de pagos: Combina información de Pagos, Socios y Planes para reportes financieros.

Socios Vigentes y No Vigentes: Vistas estratégicas que filtran usuarios basándose en la fecha actual y el estado de su última suscripción, facilitando el control de acceso.

Algoritmo 8: Vista v_socios_vigentes

```
CREATE VIEW v_socios_vigentes AS
WITH ult AS (
SELECT id_socio, MAX(fecha_fin) AS fecha_fin_max
FROM Suscripcion GROUP BY id_socio
)
SELECT s.RUT, s.nombre, p.nombre_plan, sus.fecha_fin
FROM ult
JOIN Suscripcion sus ON sus.id_socio = ult.id_socio
AND sus.fecha_fin = ult.fecha_fin_max
JOIN Socio s ON s.id_socio = sus.id_socio
WHERE sus.estado_sus = 'activa'
AND DATE(sus.fecha_fin) >= DATE('now','localtime');
```

6.2. Carga de Datos Dinámica

En lugar de utilizar scripts SQL estáticos con fechas fijas (que quedan obsoletos rápidamente), el sistema implementa un módulo de **Inyección de Datos Dinámicos** (Smart Seeding).

Este algoritmo, ejecutado al iniciar la aplicación, calcula fechas relativas al momento de la ejecución (ayer, hoy, mañana). Esto asegura que, sin importar la fecha en que se evalúe el proyecto, el Dashboard siempre muestre reportes de "los últimos 7 díasçon datos relevantes y las validaciones de vigencia funcionen correctamente con fechas actuales.

7. Desarrollo de Funcionalidades Básicas e Interfaz

Las funcionalidades CRUD (Create, Read, Update, Delete) constituyen la base de la interacción con el sistema. En GymLite, estas operaciones están reforzadas por validaciones de integridad en el backend.

7.1. Validación de las operaciones CRUD

I. CREATE: Inserción de nuevos socios con validación de unicidad de RUT.

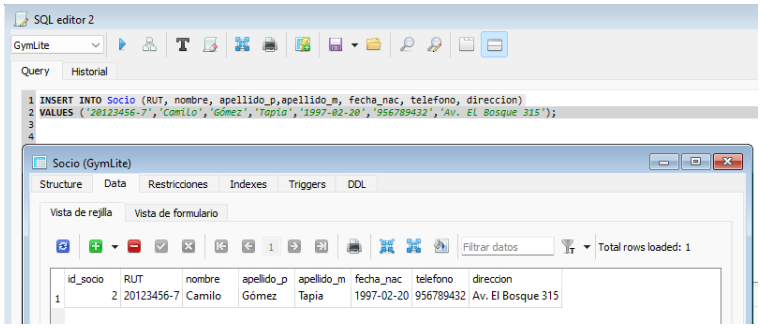

Figura 10: Formulario de registro de socio.

II. READ: Listado y filtrado de socios y estados de cuenta.

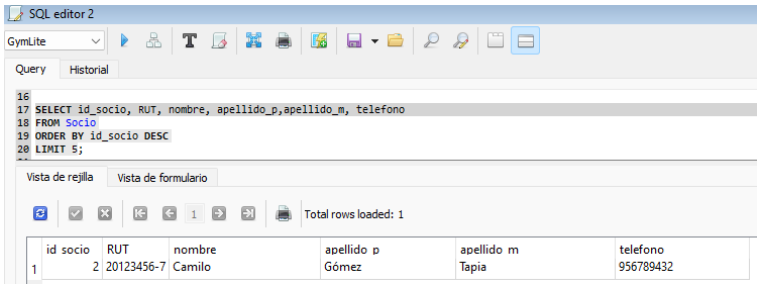

Figura 11: Vista de consulta de datos.

III. UPDATE: Modificación de datos de contacto o renovación de planes.

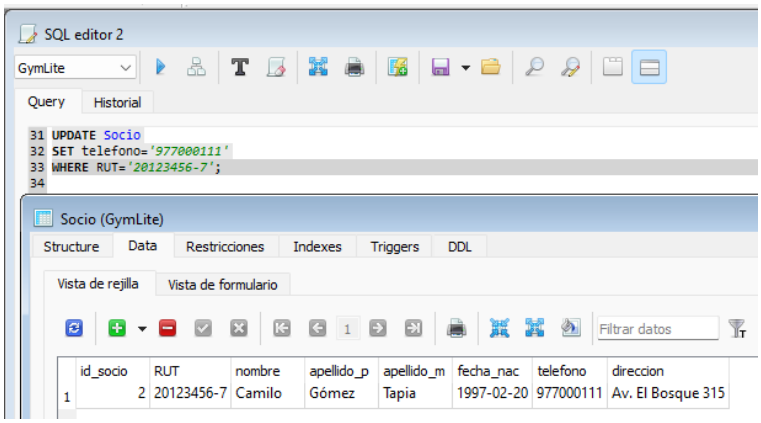

Figura 12: Actualización de registro existente.

IV. DELETE (Borrado Lógico): Esta operación ya no elimina físicamente el registro (lo que perdería el historial de pagos). En su lugar, realiza una actualización de estado (`UPDATE Socio SET activo=0`).

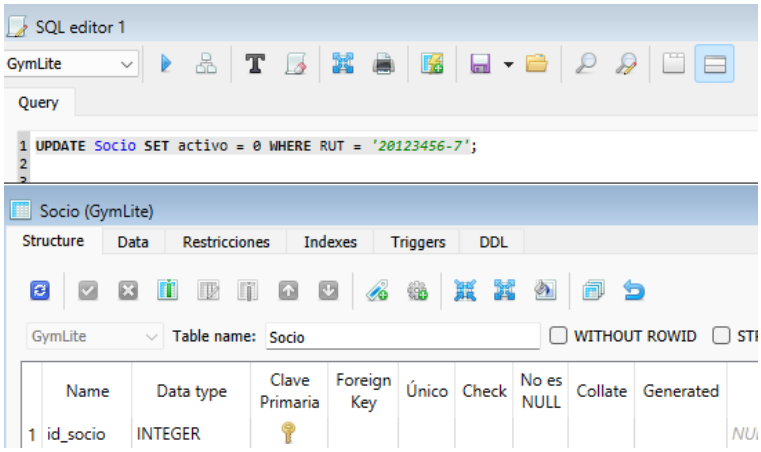

Figura 13: Acción de eliminación que ejecuta un Soft Delete.

8. Interfaz Gráfica de Usuario (GUI)

La interfaz del sistema *GymLite* ha sido desarrollada utilizando la librería `Streamlit`, priorizando la usabilidad (RNF-01) y la eficiencia en la navegación. El diseño implementa un esquema de colores oscuro (*Dark Mode*) para reducir la fatiga visual y resaltar las alertas críticas mediante contrastes de color semánticos (verde para éxito, rojo para errores o alertas).

A continuación, se detallan los módulos principales del sistema:

8.1. Dashboard Gerencial

El panel de control principal actúa como el centro de inteligencia del negocio. A diferencia de reportes estáticos, este módulo se alimenta de consultas SQL dinámicas que calculan indicadores clave de rendimiento (KPIs) en tiempo real.

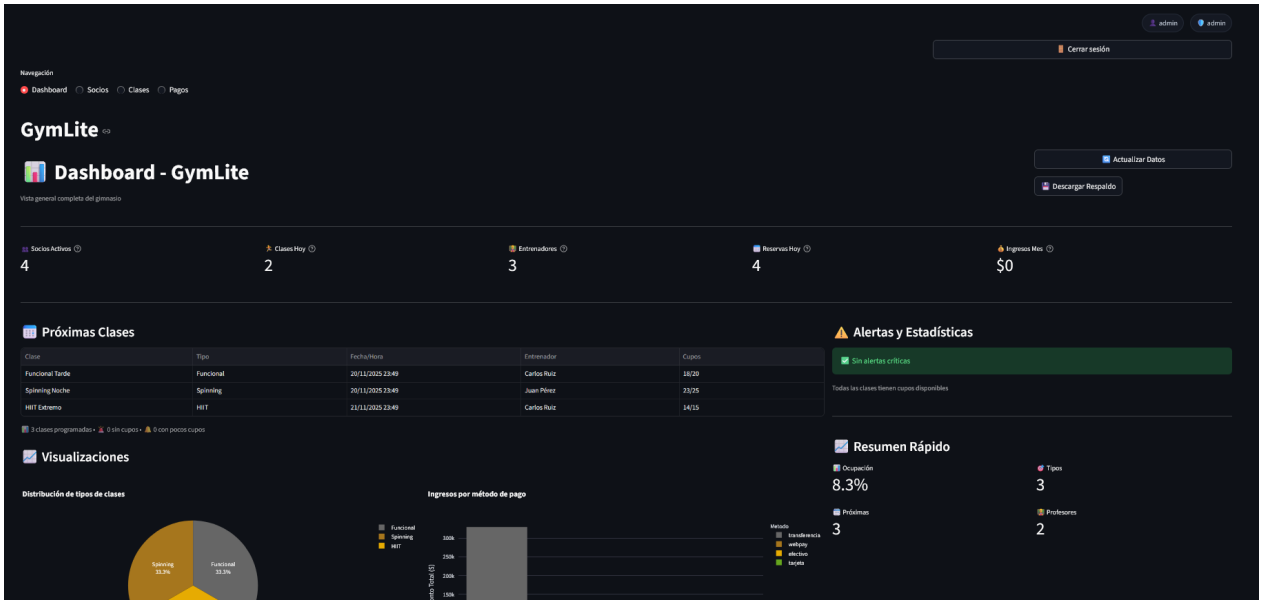

Figura 14: Vista superior del Dashboard con métricas clave (Socios Activos, Ingresos, Reservas) y alertas de sistema.

Además de los indicadores numéricos, el Dashboard incluye visualizaciones gráficas interactivas generadas con la librería `Plotly`, permitiendo analizar la evolución de reservas y la distribución de ingresos por método de pago durante los últimos 7 días.

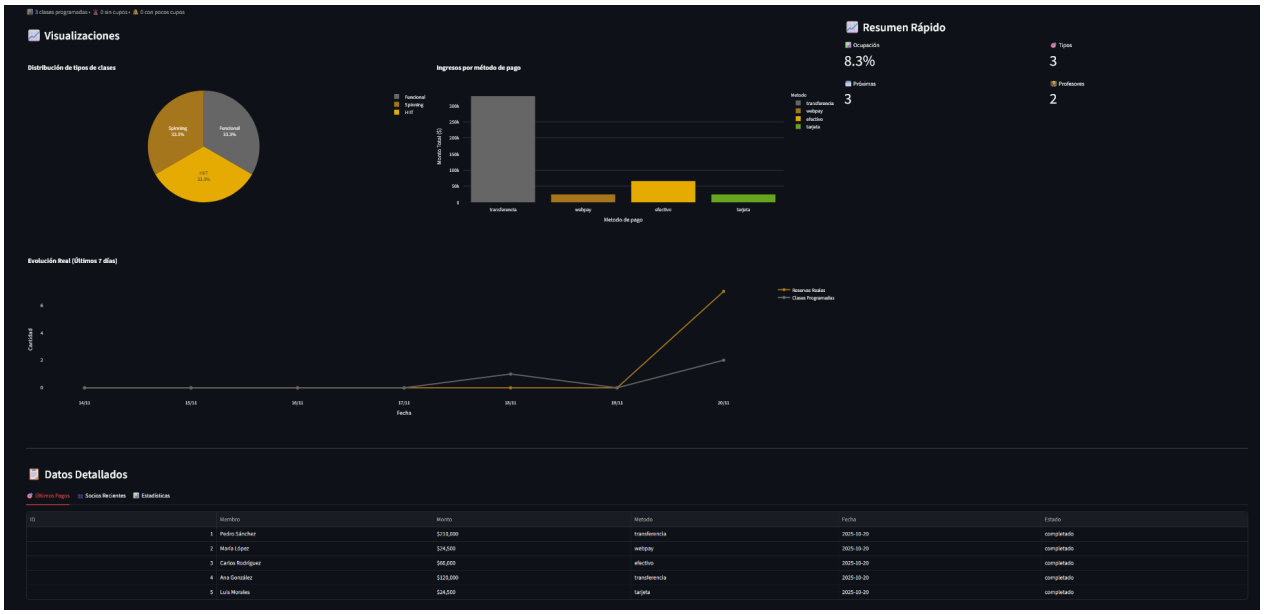

Figura 15: Sección analítica del Dashboard mostrando la evolución temporal de reservas y clases (datos dinámicos).

8.2. Módulo de Gestión de Socios

Este módulo centraliza todas las operaciones relacionadas con los miembros del gimnasio. La vista principal ofrece un listado paginado de todos los socios activos, permitiendo una visualización rápida de su estado.

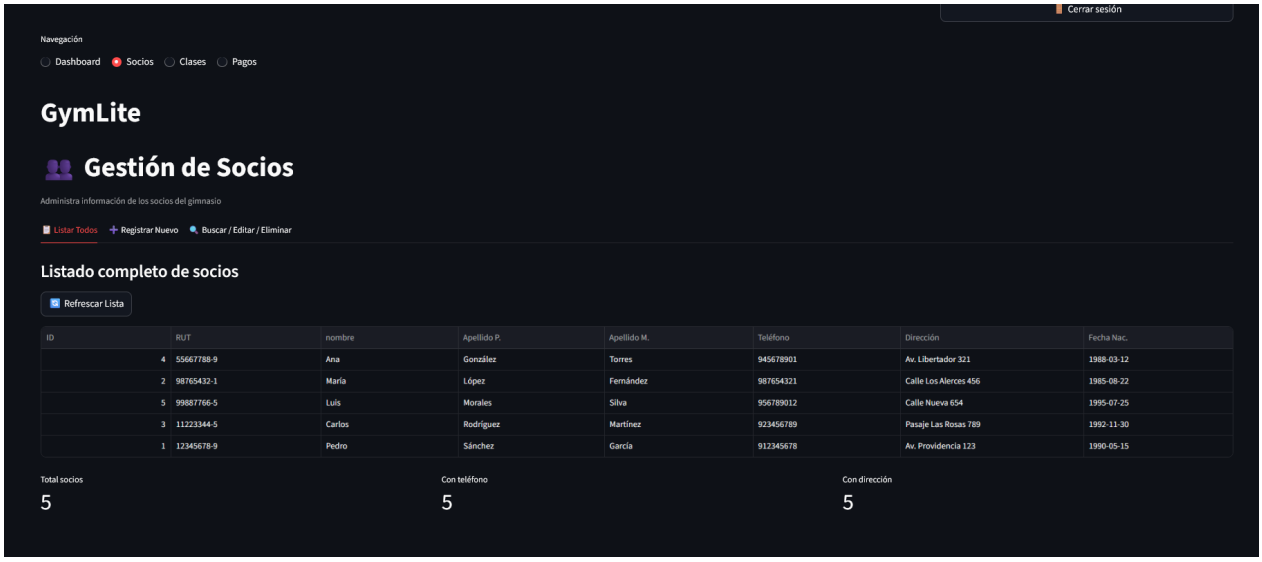

Figura 16: Listado general de socios con información de contacto y estado.

8.2.1. Registro y Mantenimiento

El formulario de registro implementa validaciones en el lado del cliente (frontend) antes de enviar los datos a la base de datos, asegurando que campos críticos como el RUT o la fecha de nacimiento cumplan con el formato esperado.

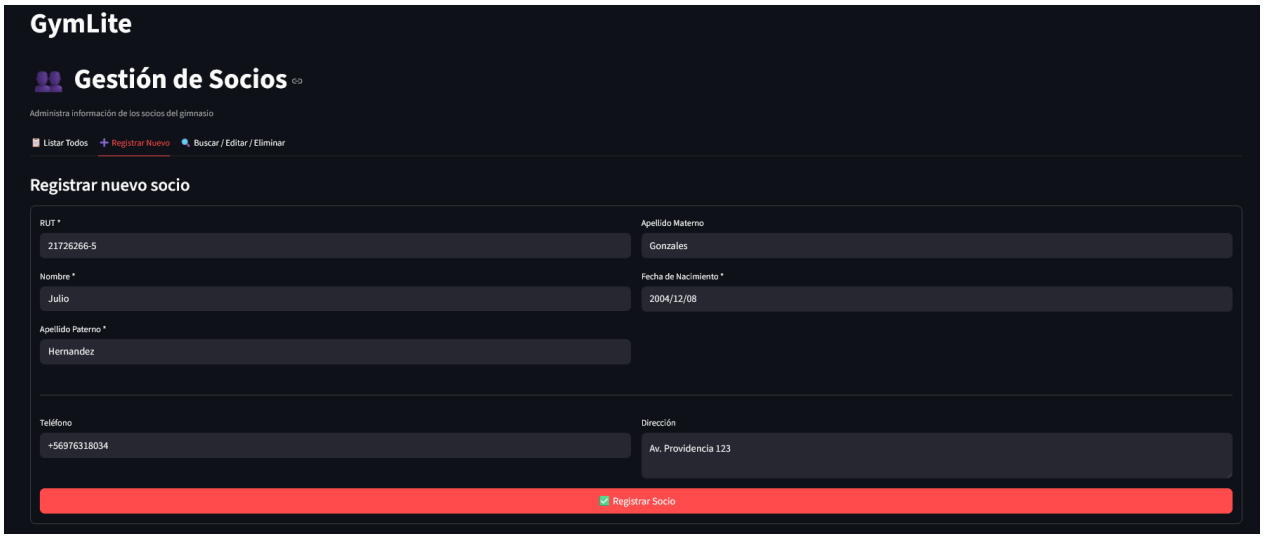

Figura 17: Formulario de registro de nuevos socios con validación de campos obligatorios.

Para la modificación de datos, se incluye una funcionalidad de búsqueda por RUT que carga automáticamente la información del socio, permitiendo editar detalles de contacto o ejecutar la baja lógica (Soft Delete) del sistema.

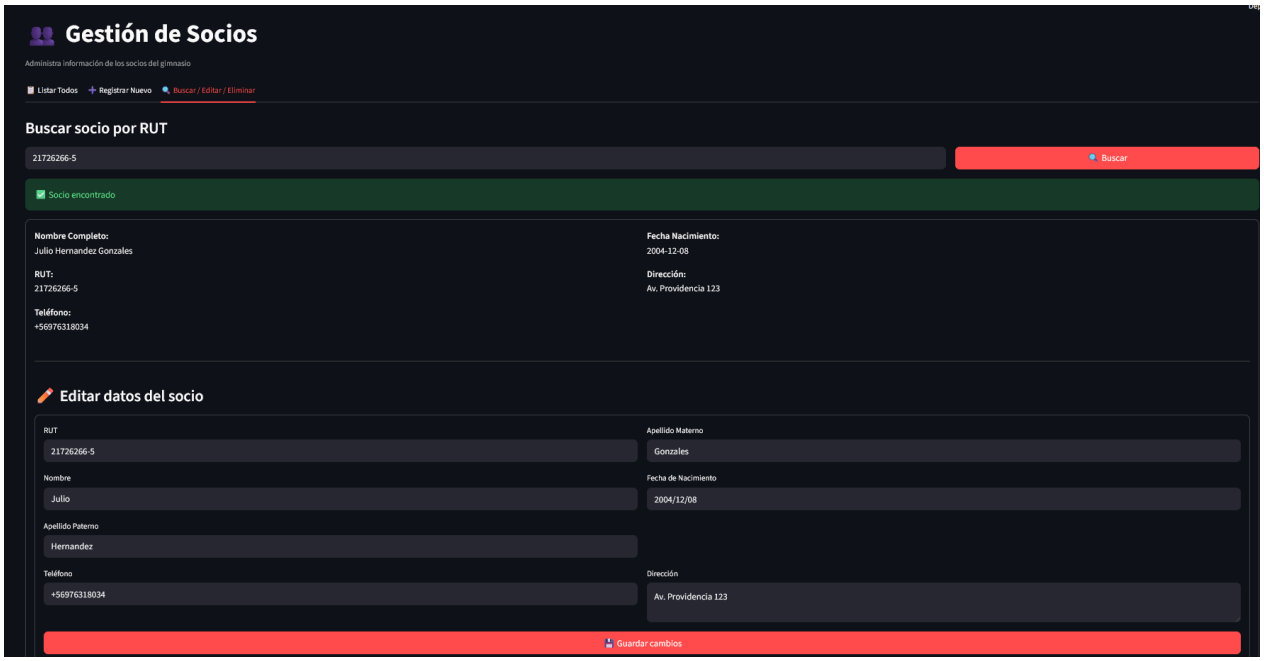

Figura 18: Módulo de búsqueda y edición de datos de socios.

8.3. Gestión de Reservas y Clases

El sistema de reservas es el punto crítico de la operación diaria. La interfaz permite visualizar las clases disponibles, mostrando en tiempo real el cupo restante (calculado mediante: $Cupo_{max} - Reservas_{confirmadas}$). Al intentar realizar una reserva, el sistema captura las excepciones lanzadas por los **Triggers** de la base de datos (como sobrecupo o morosidad) y las presenta al usuario como mensajes de error claros y descriptivos.

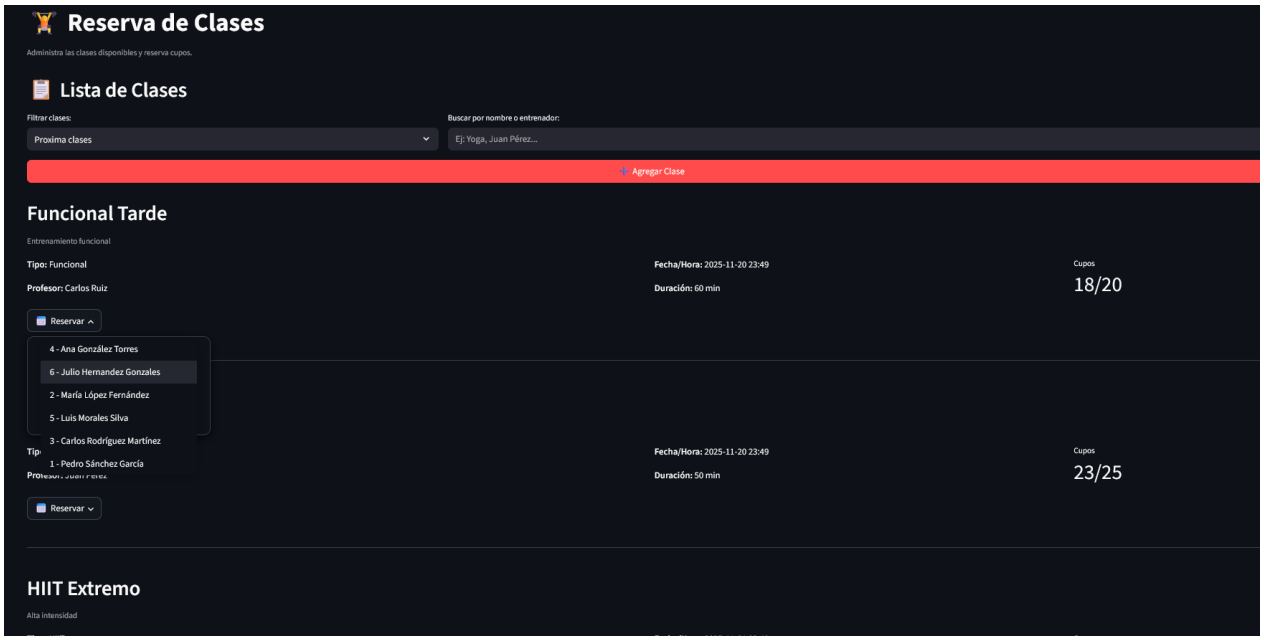

Figura 19: Interfaz de reserva de clases. Se destaca el indicador de cupos disponibles y el selector de socios.

8.4. Módulo de Pagos y Caja

Esta sección facilita el registro de transacciones financieras. El sistema está diseñado para prevenir inconsistencias: al seleccionar un socio, verifica automáticamente si posee una suscripción activa a la cual asociar el pago. Si no existe, el sistema sugiere la creación de una nueva suscripción antes de procesar el cobro, garantizando la integridad referencial (RNF-03).

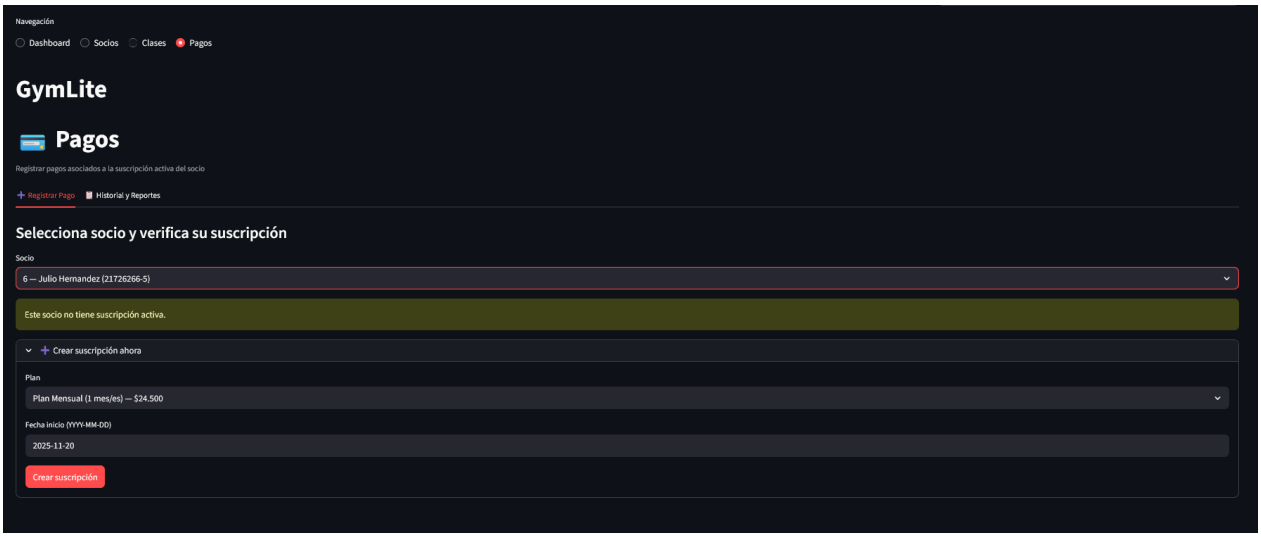

Figura 20: Módulo de registro de pagos, mostrando la asociación automática con la suscripción vigente.

A. Manual de Instalación y Despliegue

Para garantizar la correcta ejecución del sistema y resolver las observaciones sobre la configuración del entorno, se detalla el procedimiento de instalación automática.

A.1. Requisitos Previos

- Sistema Operativo: Windows 10/11, macOS o Linux.
- Python 3.10 o superior instalado.

A.2. Instalación de Dependencias

El proyecto incluye un archivo `requirements.txt` con las versiones exactas de las librerías necesarias.

1. Abra una terminal en la carpeta raíz del proyecto.
2. Ejecute el comando:

```
pip install -r requirements.txt
```

A.3. Ejecución y Auto-Configuración

El sistema cuenta con un mecanismo de *Self-Healing*. Al ejecutarse por primera vez, detecta la ausencia de la base de datos y ejecuta automáticamente los scripts de creación de tablas, triggers y carga de datos dinámicos.

1. Ejecute el comando de inicio:

```
streamlit run app.py
```

2. El sistema abrirá automáticamente el navegador.
3. Credenciales de administrador por defecto:
   - **Usuario:** `admin`
   - **Contraseña:** `Admin1234!`

B. Conclusión

El desarrollo del sistema *GymLite* ha permitido la implementación exitosa de una solución tecnológica integral para la gestión deportiva, superando las limitaciones de los procesos manuales y fragmentados identificados en la definición del problema.

Más allá del cumplimiento de los requisitos funcionales básicos (CRUD), el valor técnico del proyecto reside en las decisiones arquitectónicas adoptadas para garantizar la **integridad y confiabilidad** de la información:

- **Integridad Referencial Avanzada:** La sustitución del borrado físico por un mecanismo de **Borrado Lógico (Soft Delete)** asegura la persistencia histórica de los datos financieros, permitiendo auditorías futuras sin comprometer la consistencia de la base de datos.
- **Lógica de Negocio en Capa de Datos:** La implementación de **Triggers** para el control de aforo y validación de vigencias traslada reglas críticas de negocio directamente al motor de base de datos. Esto blinda al sistema contra errores de concurrencia (Race Conditions) que una validación únicamente a nivel de aplicación no podría garantizar.
- **Experiencia de Usuario y Despliegue:** La incorporación de mecanismos de *Smart Seeding* (carga dinámica de datos) y la automatización del despliegue facilitan la adoptabilidad del software, reduciendo la fricción en la instalación y permitiendo visualizar métricas de negocio en tiempo real desde el primer uso.

En definitiva, el sistema entregado no solo satisface la necesidad operativa de centralizar la información, sino que establece una base escalable, segura y mantenible, alineada con los estándares de ingeniería de software y bases de datos modernas.