# SQL - CODERHOUSE

PRIMER PRE ENTREGA DEL TRABAJO FINAL

Alumno Ignacio Miller

# TEMÁTICA: GIMNASIO

***Descripción:***

Este gimnasio necesita una base de datos para gestionar sus clientes, entrenadores, pagos y control de accesos. Hoy en día su administración es manual, lo que genera errores en el control de asistencia y cobros.

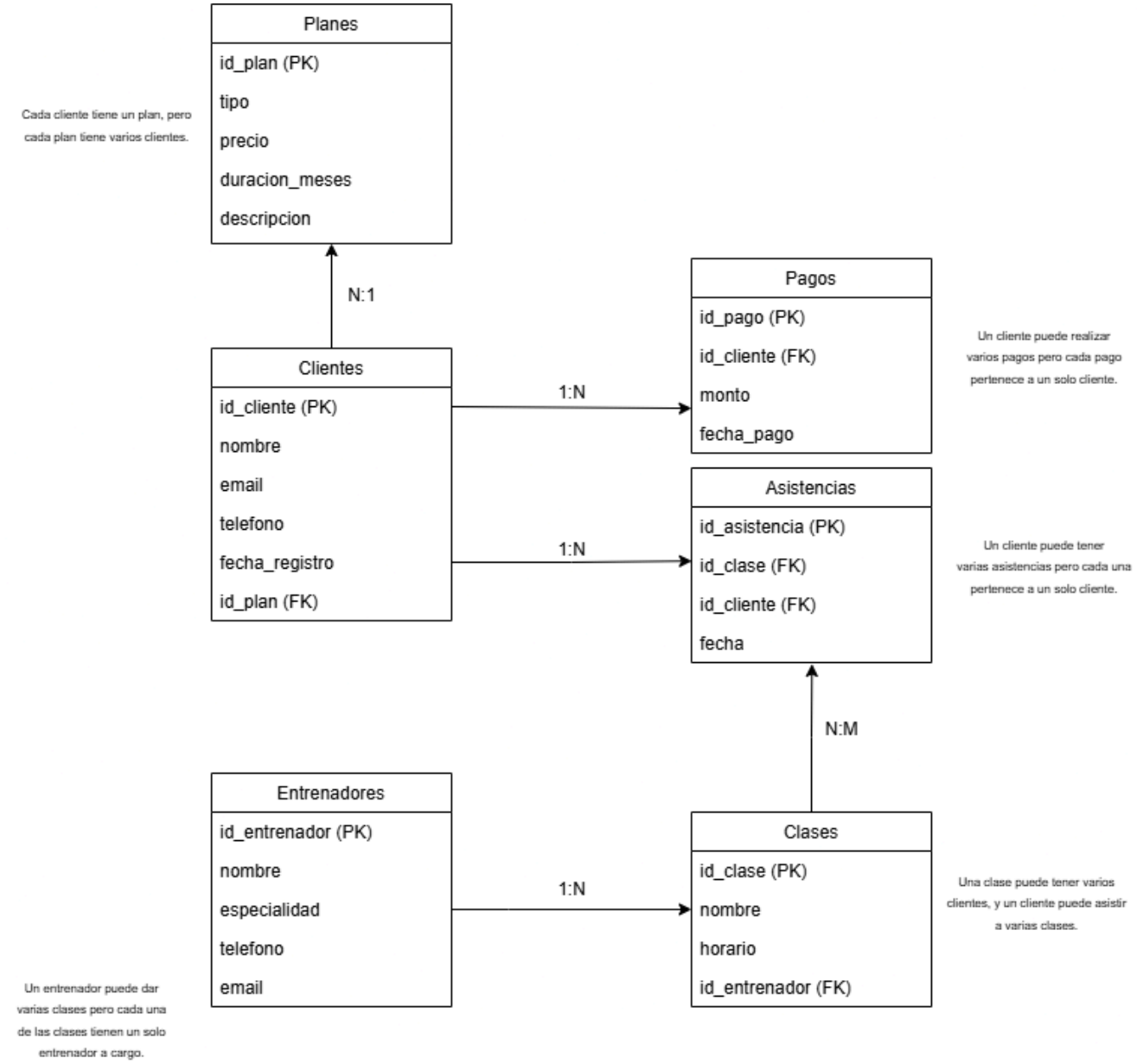
***Objetivo:***

Automatizar la gestión de clientes y pagos, asegurando un registro preciso de la información y mejorando la administración del gimnasio.

***Entidades y Relaciones:***

1. Clientes (id_cliente, nombre, email, telefono, fecha_registro, id_plan).
2. Entrenadores (id_entrenador, nombre, especialidad, telefono, email).
3. Clases (id_clase, nombre, horario, id_entrenador).
4. Planes (id_plan, tipo, precio, duracion_meses, descripcion).
5. Pagos (id_pago, id_cliente, monto, fecha_pago).
6. Asistencias (id_asistencia, id_cliente, id_clase, fecha_asistencia).

## *Diagrama Entidad-Relación:*



***Código SQL:*** GimnasioMillerSQL

***Adjunto de igual forma el código para evitar cualquier problema:***

```
-- Creación de la base de datos
CREATE DATABASE GimnasioMiller;
USE GimnasioMiller;

-- Tabla de Planes
CREATE TABLE Planes (
    id_plan INT PRIMARY KEY,
    tipo VARCHAR(50) NOT NULL,
    precio DECIMAL(10,2) NOT NULL,
    duracion_meses INT NOT NULL
);

-- Tabla de Clientes
CREATE TABLE Clientes (
    id_cliente INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    telefono VARCHAR(15),
    fecha_registro DATE NOT NULL,
    id_plan INT,
    FOREIGN KEY (id_plan) REFERENCES Planes(id_plan)
);

-- Tabla de Entrenadores
CREATE TABLE Entrenadores (
    id_entrenador INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(50) NOT NULL,
    especialidad VARCHAR(100),
    telefono VARCHAR(15),
    email VARCHAR(100) UNIQUE
);

-- Tabla de Clases
CREATE TABLE Clases (
    id_clase INT AUTO_INCREMENT PRIMARY KEY,
    nombre VARCHAR(50) NOT NULL,
    horario TIME NOT NULL,
    id_entrenador INT,
    FOREIGN KEY (id_entrenador) REFERENCES Entrenadores(id_entrenador)
);
```

```
-- Tabla de Pagos
CREATE TABLE Pagos (
    id_pago INT AUTO_INCREMENT PRIMARY KEY,
    id_cliente INT,
    monto DECIMAL(10,2) NOT NULL,
    fecha_pago DATE NOT NULL,
    FOREIGN KEY (id_cliente) REFERENCES Clientes(id_cliente)
);

-- Tabla de Asistencias
CREATE TABLE Asistencias (
    id_asistencia INT AUTO_INCREMENT PRIMARY KEY,
    id_cliente INT NOT NULL,
    id_clase INT NOT NULL,
    fecha_asistencia DATE NOT NULL DEFAULT (CURRENT_DATE),
        FOREIGN KEY (id_cliente) REFERENCES Clientes(id_cliente) ON DELETE
CASCADE,
         FOREIGN KEY (id_clase) REFERENCES Clases(id_clase) ON DELETE
CASCADE
);
```

1. ***Listado de Vistas:***

1.1 **Clientes Activos**: El objetivo de esta vista es proporcionar una lista con los clientes que tengan un plan activo. Tablas involucradas: Clientes, Planes.

```
CREATE OR REPLACE VIEW clientes_activos AS
SELECT c.id_cliente, c.nombre, c.email, c.telefono, pl.tipo AS tipo_plan
FROM Clientes c
JOIN Planes pl ON c.id_plan = pl.id_plan
WHERE pl.duracion_meses > 0;

SELECT * FROM clientes_activos;
```

1.2 **Asistencias por Clase**: El objetivo de esta vista será mostrar la cantidad de asistencias registradas por tipo de clase. Tablas involucradas: Asistencias, Clases, Cliente_Asistencia.

```
CREATE OR REPLACE VIEW asistencia_por_clase AS
SELECT cl.id_clase, cl.nombre AS nombre_clase, COUNT(ca.id_cliente) AS
total_asistencias
FROM Clases cl
LEFT JOIN Cliente_Asistencia ca ON cl.id_clase = ca.id_clase
GROUP BY cl.id_clase, cl.nombre;
```

2. ***Listado de Funciones:***

2.1 ***Tipo de plan de un cliente***: Su objetivo es devolver el tipo de plan que tiene el cliente ingresado por su ID. Tablas involucradas: Planes, Clientes.

```
-- Función para saber qué plan tiene un cliente en específico ingresando su
id_cliente
DELIMITER $$
CREATE FUNCTION obtener_plan_cliente(p_id_cliente INT)
RETURNS VARCHAR(50) DETERMINISTIC
BEGIN
    DECLARE v_tipo VARCHAR(50);

    SELECT p.tipo
    INTO v_tipo
    FROM Clientes c
    JOIN Planes p ON c.id_plan = p.id_plan
    WHERE c.id_cliente = p_id_cliente
```

```
    LIMIT 1;

    RETURN v_tipo;
END $$
DELIMITER ;

SELECT obtener_plan_cliente(1);
```

2.2 ***Contar clientes por tipo de plan***: El objetivo de esta función es contar la cantidad de clientes inscriptos a cada tipo de plan en el gimnasio. Tablas involucradas: Planes, Clientes.

```
-- Función para contar la cantidad de clientes por ID de plan
DELIMITER $$
CREATE FUNCTION cantidad_clientes_por_plan(p_id_plan INT)
RETURNS INT DETERMINISTIC
BEGIN
    DECLARE v_cantidad INT DEFAULT 0;

    SELECT COUNT(*)
    INTO v_cantidad
    FROM Clientes
    WHERE id_plan = p_id_plan;

    RETURN v_cantidad;
END $$

DELIMITER ;

SELECT cantidad_clientes_por_plan(2);
```

2.3 ***Calcular ingresos en un período de tiempo***: El objetivo de esta función es calcular el monto total de ingresos del gimnasio entre dos fechas ingresadas manualmente. Tablas involucradas: Pagos.

```
-- 2.3 Función para calcular total de ingresos en un periodo de tiempo
DELIMITER $$
CREATE FUNCTION total_ingresos(fecha_inicio DATE, fecha_fin DATE)
RETURNS DECIMAL(10,2) DETERMINISTIC
BEGIN
    DECLARE total DECIMAL(10,2);
    SELECT IFNULL(SUM(monto), 0) INTO total FROM Pagos
```

```
    WHERE fecha_pago BETWEEN fecha_inicio AND fecha_fin;
    RETURN total;
END $$

DELIMITER ;

SELECT total_ingresos('2025-01-21', NOW());
```

### 3. *Listado de Stored Procedures:*

3.1 ***Registrar Pagos***: Su objetivo es registrar un nuevo pago en la base de datos. Tablas involucradas: Pagos.

```
-- Stored Procedure para registrar pagos nuevos
DELIMITER $$
CREATE PROCEDURE sp_registrar_pagos(IN id_cliente INT, IN monto DECIMAL(10,2), IN fecha_pago DATE)
BEGIN
    INSERT INTO Pagos (id_cliente, monto, fecha_pago) VALUES (id_cliente, monto, fecha_pago);
END $$
DELIMITER ;

call sp_registrar_pagos(2, 500000.00, '2025-03-02');
```

3.2 ***Registrar Asistencias***: Su objetivo es registrar la asistencia de un cliente a una clase. Tablas involucradas: Clientes, Asistencias.

```
-- Stored Procedure para registrar asistencias
DELIMITER $$
CREATE PROCEDURE sp_registrar_asistencia(
    IN p_id_cliente INT,
    IN p_id_clase INT,
    IN p_fecha_asistencia DATE
)
BEGIN
    -- Verificar si el cliente existe
    IF NOT EXISTS (SELECT 1 FROM Clientes WHERE id_cliente = p_id_cliente) THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Error: Cliente no encontrado';
    END IF;
```

```
    -- Verificar si la clase existe
    IF NOT EXISTS (SELECT 1 FROM Clases WHERE id_clase = p_id_clase) THEN
            SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Error: Clase no
encontrada';
    END IF;

    -- Insertar la asistencia en la tabla
    INSERT INTO Asistencias (id_cliente, id_clase, fecha_asistencia)
    VALUES (p_id_cliente, p_id_clase, p_fecha_asistencia);
END $$

DELIMITER ;

call sp_registrar_asistencia(1, 2, CURDATE());
```

### *4. Triggers:*

4.1 ***Trigger para ver los logs de registros de asistencias***: Este trigger se ejecutará después (AFTER INSERT) de registrar una nueva asistencia en la base de datos, su objetivo es crear una tabla con los logs de las asistencias que se vayan registrando cuando un cliente asiste a una clase. Tablas involucradas: Clientes, Asistencias.

```
-- Creación de la tabla para los logs
CREATE TABLE IF NOT EXISTS logs_asistencias (
    id_log INT AUTO_INCREMENT PRIMARY KEY,
    id_cliente INT,
    id_clase INT,
    fecha_asistencia DATE,
    mensaje VARCHAR(255)
);

-- Creación del trigger after insert para las asistencias
DELIMITER $$
CREATE TRIGGER after_insert_asistencia
AFTER INSERT ON asistencias
FOR EACH ROW
BEGIN
    INSERT INTO logs_asistencias (id_cliente, id_clase, fecha_asistencia, mensaje)
        VALUES (NEW.id_cliente, NEW.id_clase, NEW.fecha_asistencia, 'Nueva
asistencia registrada');
END $$
DELIMITER ;
```

5. ***Inserción de datos para las tablas.***

```
-- 5 INSERCIÓN DE DATOS PARA LAS TABLAS
INSERT INTO Planes (id_plan, tipo, duracion_meses, precio) VALUES
(1, 'Mensual', 1, 30000.00),
(2, 'Trimestral', 3, 85000.00),
(3, 'Semestral', 6, 150000.00),
(4, 'Anual', 12, 280000.00),
(5, 'Expirado', 0, 0);

INSERT INTO Clientes (id_cliente, nombre, email, telefono, fecha_registro, id_plan)
VALUES
(1, 'Juan Pérez', 'juan.perez@gmail.com', '3513837458','2025-02-01', 1),
(2, 'María González', 'maria.gonzalez@gmail.com', '3513933882','2024-12-20', 2),
(3, 'Pedro Alvarado', 'pedro.alvarado@gmail.com', '3517643215','2024-07-04', 4),
(4, 'Matías Cetreno', 'matias.cetreno@gmail.com', '3518722556','2024-09-11', 3),
(5, 'Ignacio Miller', 'ignacio.miller@gmail.com', '3514944857','2023-04-30', 5);

INSERT INTO Pagos (id_pago, id_cliente, monto, fecha_pago) VALUES
(1, 3, 280000.00, '2024-09-04'),
(2, 2, 85000.00, '2025-01-20');

INSERT INTO Clases (id_clase, nombre, horario) VALUES
(1, 'Yoga', '08:00:00'),
(2, 'Crossfit', '18:00:00'),
(3, 'Musculación', '12:00:00');

INSERT INTO Asistencias (id_cliente, id_clase, fecha_asistencia) VALUES
(1, 1, '2025-02-20'),
(2, 2, '2025-02-21'),
(3, 3, '2025-02-22'),
(2, 3, '2025-02-19');

INSERT INTO Entrenadores (nombre, especialidad, telefono, email) VALUES
('Carlos Mendoza', 'Crossfit', '3511234567', 'carlos.mendoza@gmail.com'),
('Lucía Fernández', 'Yoga', '3512345678', 'lucia.fernandez@gmail.com'),
('Javier Torres', 'Musculación', '3515678901', 'javier.torres@gmail.com');
```