Proyecto Centro Deportivo

Miguel Ángel Chaparro

CODERHOUSE

Curso de Bases de Datos

Zipaquirá

2025

Descripción de la temática de la base de datos:

Mi base de datos la estructure para un centro deportivo con clientes, empleados, membresías, reservas o clases, instalaciones del centro deportivo, pagos y equipamiento.

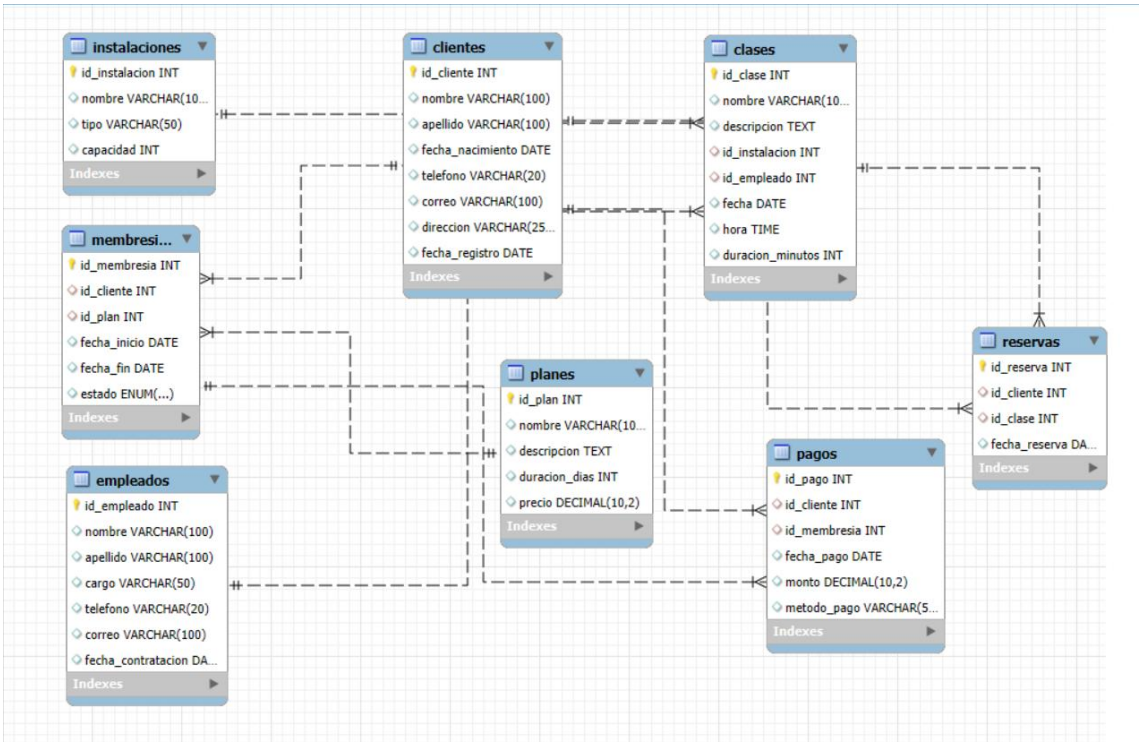
Situación problemática:

El objetivo de esta base de datos es ayudar a gestionar un centro deportivo para poder organizar y agilizar los procesos tanto internos como externos para mejorar la atención, llevar un registro actualizado de todo el negocio y mejorar la eficiencia y la eficacia en este.

Apartado Modelo de Negocio:

Se va a utilizar para un centro deportivo de tamaño mediano grande que recibe tanto a gente que le gusta hacer deporte como a deportistas de alto rendimiento, posee instalaciones para practicas varios deportes, aprender y en caso de que ya seas un profesional, entrenar con las mejores instalaciones y adecuaciones mediante membresías e inscripciones, para mejorar y rendir mejor en todos los deportes.

Diagrama de Entidad – Relación:



Listado de tablas:

Tabla Clientes:

id_cliente INT AUTO_INCREMENT PRIMARY KEY,

nombre VARCHAR(100)

apellido VARCHAR(100)

fecha_nacimiento DATE

telefono VARCHAR(20)

correo VARCHAR(100)

direccion VARCHAR(255)

fecha_registro DATE

Tabla Empleados:

id_empleado INT AUTO_INCREMENT PRIMARY KEY

nombre VARCHAR(100)

apellido VARCHAR(100)

cargo VARCHAR(50)

telefono VARCHAR(20)

correo VARCHAR(100)

fecha_contratacion DATE

Tabla Planes:

id_plan INT AUTO_INCREMENT PRIMARY KEY,

nombre VARCHAR(100),

descripcion TEXT,

duracion_dias INT,

precio DECIMAL(10,2)

Tabla Membresías:

id_membresia INT AUTO_INCREMENT PRIMARY KEY,

id_cliente INT,

id_plan INT,

fecha_inicio DATE,

fecha_fin DATE,

estado ENUM('Activa', 'Inactiva'),

FOREIGN KEY (id_cliente) REFERENCES Clientes(id_cliente),

FOREIGN KEY (id_plan) REFERENCES Planes(id_plan)

Tabla Instalaciones:

id_instalacion INT AUTO_INCREMENT PRIMARY KEY,

nombre VARCHAR(100),

tipo VARCHAR(50),

capacidad INT

Tabla Clases:

id_clase INT AUTO_INCREMENT PRIMARY KEY,

nombre VARCHAR(100),

descripcion TEXT,

id_instalacion INT,

id_empleado INT,

fecha DATE,

hora TIME,

duracion_minutos INT,

FOREIGN KEY (id_instalacion) REFERENCES Instalaciones(id_instalacion),

FOREIGN KEY (id_empleado) REFERENCES Empleados(id_empleado)

Tabla Reservas:

id_reserva INT AUTO_INCREMENT PRIMARY KEY,

id_cliente INT,

id_clase INT,

fecha_reserva DATE,

FOREIGN KEY (id_cliente) REFERENCES Clientes(id_cliente),

FOREIGN KEY (id_clase) REFERENCES Clases(id_clase)

Tabla Pagos:

id_pago INT AUTO_INCREMENT PRIMARY KEY,

id_cliente INT,

id_membresia INT,

fecha_pago DATE,

monto DECIMAL(10,2),

metodo_pago VARCHAR(50),

FOREIGN KEY (id_cliente) REFERENCES Clientes(id_cliente),

FOREIGN KEY (id_membresia) REFERENCES Membresias(id_membresia)

Link del archivo SQL que contiene el script de la base de datos y el modelo entidad relación generado:

https://github.com/Miguesner/Proyecto-Centro-Deportivo

Formato:

Se cargaron dos archivos de cada uno, uno con el nombre inicial y otro con el nombre inicial y mi apellido según lo solicitado en las instrucciones de entrega.

Entrega 2

Listado de Vistas:

1. vista_clientes_membresias_activas

Descripción:

Muestra los clientes con membresías activas, incluyendo el nombre del plan, fecha de inicio y fin.

Objetivo:

Consultar fácilmente qué clientes tienen una membresía activa, útil para validaciones y reportes administrativos.

Tablas involucradas:

- Clientes
- Membresias
- Planes

Vista SQL:

CREATE VIEW vista_clientes_membresias_activas AS

SELECT

c.id_cliente,

CONCAT(c.nombre, ' ', c.apellido) AS nombre_completo,

p.nombre AS plan,

m.fecha_inicio,

m.fecha_fin

FROM Membresias m

JOIN Clientes c ON m.id_cliente = c.id_cliente

JOIN Planes p ON m.id_plan = p.id_plan

WHERE m.estado = 'Activa';

2. vista_clases_programadas

Descripción:

Lista todas las clases programadas con detalles del instructor, instalación y horario.

Objetivo:

Ver la programación de clases con información relevante para los asistentes o el personal de coordinación.

Tablas involucradas:

- Clases
- Instalaciones
- Empleados

Vista SQL:

CREATE VIEW vista_clases_programadas AS

SELECT

cl.id_clase,

cl.nombre AS clase,

cl.descripcion,

i.nombre AS instalacion,

e.nombre AS instructor,

cl.fecha,

cl.hora,

cl.duracion_minutos

FROM Clases cl

JOIN Instalaciones i ON cl.id_instalacion = i.id_instalacion

JOIN Empleados e ON cl.id_empleado = e.id_empleado;

3. vista_reservas_clientes

Descripción:

Muestra las reservas de clases hechas por los clientes.

Objetivo:

Tener un seguimiento claro de qué clases ha reservado cada cliente.

Tablas involucradas:

- Reservas
- Clientes
- Clases

Vista SQL:

```
CREATE VIEW vista_reservas_clientes AS
SELECT
  r.id_reserva,
  CONCAT(c.nombre, ' ', c.apellido) AS cliente,
  cl.nombre AS clase,
  cl.fecha,
  cl.hora,
  r.fecha_reserva
FROM Reservas r
JOIN Clientes c ON r.id_cliente = c.id_cliente
JOIN Clases cl ON r.id_clase = cl.id_clase;
```

4. vista_pagos_realizados

Descripción:

Muestra los pagos realizados por los clientes, junto con el plan al que están asociados.

Objetivo:

Tener visibilidad financiera sobre los ingresos por cliente y plan.

Tablas involucradas:

- Pagos
- Clientes
- Membresias
- Planes

Vista SQL:

```
CREATE VIEW vista_pagos_realizados AS
SELECT
  p.id_pago,
  CONCAT(c.nombre, ' ', c.apellido) AS cliente,
  pl.nombre AS plan,
```

```
    p.fecha_pago,
    p.monto,
    p.metodo_pago
FROM Pagos p
JOIN Clientes c ON p.id_cliente = c.id_cliente
JOIN Membresias m ON p.id_membresia = m.id_membresia
JOIN Planes pl ON m.id_plan = pl.id_plan;
```

5. vista_empleados_clases

Descripción:

Lista de empleados con las clases que imparten y la instalación asignada.

Objetivo:

Revisar qué empleados están asignados a qué clases y en qué lugar.

Tablas involucradas:

- Empleados
- Clases
- Instalaciones

Vista SQL:

```
CREATE VIEW vista_empleados_clases AS
SELECT
    e.id_empleado,
    CONCAT(e.nombre, ' ', e.apellido) AS empleado,
    cl.nombre AS clase,
    cl.fecha,
    cl.hora,
    i.nombre AS instalacion
FROM Clases cl
JOIN Empleados e ON cl.id_empleado = e.id_empleado
JOIN Instalaciones i ON cl.id_instalacion = i.id_instalacion;
```

6. vista_membresias_vencidas

Descripción:

Clientes cuyas membresías ya han expirado.

Objetivo:

Detectar clientes que deben renovar su membresía o hacer seguimiento comercial.

Tablas involucradas:

- Clientes
- Membresias
- Planes

Vista SQL:

```
CREATE VIEW vista_membresias_vencidas AS
SELECT
   c.id_cliente,
   CONCAT(c.nombre, ' ', c.apellido) AS cliente,
   p.nombre AS plan,
   m.fecha_fin
FROM Membresias m
JOIN Clientes c ON m.id_cliente = c.id_cliente
JOIN Planes p ON m.id_plan = p.id_plan
WHERE m.fecha_fin < CURDATE()
  AND m.estado = 'Inactiva';
```

Listado de Funciones:

1. fn_calcular_edad_cliente(idCliente INT)

Descripción:

Calcula la edad actual de un cliente a partir de su fecha de nacimiento.

Objetivo:

Obtener dinámicamente la edad del cliente para reportes o segmentaciones (jóvenes, adultos mayores, etc.).

Tablas:

- Clientes

Función SQL:

```
CREATE FUNCTION fn_calcular_edad_cliente(idCliente INT)
RETURNS INT
DETERMINISTIC
BEGIN
   DECLARE edad INT;
```

```
    SELECT TIMESTAMPDIFF(YEAR, fecha_nacimiento, CURDATE())
    INTO edad
    FROM Clientes
    WHERE id_cliente = idCliente;
    RETURN edad;
END;
```

2. fn_membresia_vigente(idCliente INT)

Descripción:

Devuelve 1 si el cliente tiene una membresía activa y vigente, 0 en caso contrario.

Objetivo:

Validar rápidamente si un cliente puede reservar clases o acceder a las instalaciones.

Tablas:

- Membresias

Función SQL:

```
CREATE FUNCTION fn_membresia_vigente(idCliente INT)
RETURNS BOOLEAN
DETERMINISTIC
BEGIN
    DECLARE vigente BOOLEAN;
    SELECT COUNT(*) > 0 INTO vigente
    FROM Membresias
    WHERE id_cliente = idCliente
      AND estado = 'Activa'
      AND CURDATE() BETWEEN fecha_inicio AND fecha_fin;
    RETURN vigente;
END;
```

3. fn_total_pagado_cliente(idCliente INT)

Descripción:

Suma todos los pagos realizados por un cliente.

Objetivo:

Obtener cuánto ha pagado un cliente históricamente para métricas de ingresos o fidelidad.

Tablas:

- Pagos

Función SQL:

```
CREATE FUNCTION fn_total_pagado_cliente(idCliente INT)
RETURNS DECIMAL(10,2)
DETERMINISTIC
BEGIN
  DECLARE total DECIMAL(10,2);
  SELECT COALESCE(SUM(monto), 0) INTO total
  FROM Pagos
  WHERE id_cliente = idCliente;
  RETURN total;
END;
```

4. fn_reservas_cliente_mes(idCliente INT, anio INT, mes INT)

Descripción:

Devuelve el número de reservas que ha hecho un cliente en un mes y año específicos.

Objetivo:

Analizar la participación mensual de los clientes (útil para promociones, métricas de uso).

Tablas:

- Reservas

Función SQL:

```
CREATE FUNCTION fn_reservas_cliente_mes(idCliente INT, anio INT, mes INT)
RETURNS INT
DETERMINISTIC
BEGIN
  DECLARE total INT;
  SELECT COUNT(*) INTO total
  FROM Reservas
  WHERE id_cliente = idCliente
    AND YEAR(fecha_reserva) = anio
    AND MONTH(fecha_reserva) = mes;
```

```
    RETURN total;
END;
```

5. fn_capacidad_disponible_clase(idClase INT)

Descripción:

Devuelve el número de cupos disponibles en una clase programada.

Objetivo:

Evitar sobrecupo en reservas y controlar la capacidad de las instalaciones.

Tablas:

- Clases
- Reservas
- Instalaciones

Función SQL:

```
CREATE FUNCTION fn_capacidad_disponible_clase(idClase INT)
RETURNS INT
DETERMINISTIC
BEGIN
    DECLARE capacidad INT;
    DECLARE reservados INT;
    DECLARE disponibles INT;

    SELECT i.capacidad INTO capacidad
    FROM Clases c
    JOIN Instalaciones i ON c.id_instalacion = i.id_instalacion
    WHERE c.id_clase = idClase;

    SELECT COUNT(*) INTO reservados
    FROM Reservas
    WHERE id_clase = idClase;

    SET disponibles = capacidad - reservados;

    RETURN disponibles;
```

END;

Listado de Procedimientos almacenados:

1. sp_registrar_cliente

Descripción:

Registra un nuevo cliente en la base de datos con todos sus datos personales.

Objetivo / Beneficio:

Centralizar y asegurar el proceso de alta de nuevos clientes, evitando errores en inserciones manuales.

Tablas involucradas:

- Clientes

Procedimiento SQL:

```
CREATE PROCEDURE sp_registrar_cliente (
    IN p_nombre VARCHAR(100),
    IN p_apellido VARCHAR(100),
    IN p_fecha_nacimiento DATE,
    IN p_telefono VARCHAR(20),
    IN p_correo VARCHAR(100),
    IN p_direccion VARCHAR(255)
)
BEGIN
    INSERT INTO Clientes (nombre, apellido, fecha_nacimiento, telefono, correo, direccion, fecha_registro)
    VALUES (p_nombre, p_apellido, p_fecha_nacimiento, p_telefono, p_correo, p_direccion, CURDATE());
END;
```

2. sp_asignar_membresia

Descripción:

Asigna una nueva membresía a un cliente, calculando la fecha fin en función de la duración del plan.

Objetivo / Beneficio:

Automatizar el proceso de inscripción a planes, asegurando coherencia en fechas y estado.

Tablas involucradas:

- Membresias

- Planes

Procedimiento SQL:

```
CREATE PROCEDURE sp_asignar_membresia (
   IN p_id_cliente INT,
   IN p_id_plan INT,
   IN p_fecha_inicio DATE
)
BEGIN
   DECLARE duracion INT;
   DECLARE fecha_fin DATE;

   SELECT duracion_dias INTO duracion FROM Planes WHERE id_plan = p_id_plan;
   SET fecha_fin = DATE_ADD(p_fecha_inicio, INTERVAL duracion DAY);

   INSERT INTO Membresias (id_cliente, id_plan, fecha_inicio, fecha_fin, estado)
   VALUES (p_id_cliente, p_id_plan, p_fecha_inicio, fecha_fin, 'Activa');
END;
```

3. sp_registrar_pago

Descripción:

Registra un nuevo pago hecho por un cliente y lo asocia con una membresía.

Objetivo / Beneficio:

Controlar de manera estructurada la entrada de pagos, dejando trazabilidad clara.

Tablas involucradas:

- Pagos

Procedimiento SQL:

```
CREATE PROCEDURE sp_registrar_pago (
   IN p_id_cliente INT,
   IN p_id_membresia INT,
   IN p_monto DECIMAL(10,2),
   IN p_metodo_pago VARCHAR(50)
)
BEGIN
```

```
    INSERT INTO Pagos (id_cliente, id_membresia, fecha_pago, monto, metodo_pago)
    VALUES (p_id_cliente, p_id_membresia, CURDATE(), p_monto, p_metodo_pago);
END;
```

4. sp_reservar_clase

Descripción:

Permite a un cliente reservar una clase, validando si aún hay cupos disponibles.

Objetivo / Beneficio:

Evitar reservas excesivas y respetar la capacidad de las instalaciones.

Tablas involucradas:

- Reservas
- Clases
- Instalaciones

Procedimiento SQL:

```
CREATE PROCEDURE sp_reservar_clase (
    IN p_id_cliente INT,
    IN p_id_clase INT
)
BEGIN
    DECLARE capacidad INT;
    DECLARE reservados INT;

    SELECT i.capacidad INTO capacidad
    FROM Clases c
    JOIN Instalaciones i ON c.id_instalacion = i.id_instalacion
    WHERE c.id_clase = p_id_clase;

    SELECT COUNT(*) INTO reservados
    FROM Reservas
    WHERE id_clase = p_id_clase;

    IF reservados < capacidad THEN
        INSERT INTO Reservas (id_cliente, id_clase, fecha_reserva)
```

```
    VALUES (p_id_cliente, p_id_clase, CURDATE());
  ELSE
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Clase completa: no hay cupos disponibles';
  END IF;
END;
```

5. sp_actualizar_estado_membresias

Descripción:

Actualiza automáticamente el estado de las membresías vencidas a "Inactiva".

Objetivo / Beneficio:

Mantener la base de datos limpia y reflejar el estado real de las membresías sin intervención manual.

Tablas involucradas:

- Membresias

Procedimiento SQL:

```
CREATE PROCEDURE sp_actualizar_estado_membresias ()
BEGIN
  UPDATE Membresias
  SET estado = 'Inactiva'
  WHERE fecha_fin < CURDATE()
   AND estado = 'Activa';
END;
```

6. sp_listado_clientes_con_deuda

Descripción:

Devuelve una lista de clientes que tienen membresía activa pero no han realizado pago en los últimos X días.

Objetivo / Beneficio:

Ayuda en tareas de cobranza y seguimiento financiero.

Tablas involucradas:

- Clientes
- Membresias
- Pagos

Procedimiento SQL:

```
CREATE PROCEDURE sp_listado_clientes_con_deuda (IN p_dias INT)
BEGIN
   SELECT c.id_cliente, CONCAT(c.nombre, ' ', c.apellido) AS nombre_completo
   FROM Clientes c
   JOIN Membresias m ON c.id_cliente = m.id_cliente
   WHERE m.estado = 'Activa'
    AND NOT EXISTS (
       SELECT 1
       FROM Pagos p
       WHERE p.id_cliente = c.id_cliente
        AND p.fecha_pago >= DATE_SUB(CURDATE(), INTERVAL p_dias DAY)
    );
END;
```

Listado de Triggers:

1. Trigger: trg_membresia_estado_automatico

- **Descripción:** Al insertar una nueva membresía, este trigger establece automáticamente su estado en 'Activa' si la fecha final es mayor o igual a la fecha actual.
- **Objetivo:** Asegura la coherencia del campo estado en la tabla Membresias sin intervención manual.
- **Tablas involucradas:** Membresias

```
DROP TRIGGER IF EXISTS trg_membresia_estado_automatico;
DELIMITER $$
CREATE TRIGGER trg_membresia_estado_automatico
BEFORE INSERT ON Membresias
FOR EACH ROW
BEGIN
   IF NEW.fecha_fin >= CURDATE() THEN
      SET NEW.estado = 'Activa';
   ELSE
      SET NEW.estado = 'Inactiva';
   END IF;
END$$
```

```
DELIMITER ;
```

2. Trigger: trg_actualizar_estado_membresia

- **Descripción:** Cambia el estado de una membresía a 'Inactiva' si la fecha de finalización es modificada a una fecha anterior a hoy.
- **Objetivo:** Mantener el campo estado sincronizado con la validez real de la membresía.
- **Tablas involucradas:** Membresias

```
DROP TRIGGER IF EXISTS trg_actualizar_estado_membresia;

DELIMITER $$

CREATE TRIGGER trg_actualizar_estado_membresia

BEFORE UPDATE ON Membresias

FOR EACH ROW

BEGIN

   IF NEW.fecha_fin < CURDATE() THEN

      SET NEW.estado = 'Inactiva';

   ELSE

      SET NEW.estado = 'Activa';

   END IF;

END$$

DELIMITER ;
```

3. Trigger: trg_pago_registro_auditoria

- **Descripción:** Al registrar un nuevo pago, guarda automáticamente un registro de auditoría en una tabla auxiliar.
- **Objetivo:** Mantener un historial de pagos para trazabilidad.
- **Tablas involucradas:** Pagos, Pagos_Auditoria (se requiere crearla)

```
-- Tabla auxiliar para auditoría

CREATE TABLE IF NOT EXISTS Pagos_Auditoria (

   id_auditoria INT AUTO_INCREMENT PRIMARY KEY,

   id_pago INT,

   id_cliente INT,

   id_membresia INT,

   fecha_pago DATETIME,

   monto DECIMAL(10,2),
```

```
    metodo_pago VARCHAR(50),
    accion VARCHAR(10),
    fecha_auditoria DATETIME DEFAULT NOW()
);
-- Trigger
DROP TRIGGER IF EXISTS trg_pago_registro_auditoria;
DELIMITER $$
CREATE TRIGGER trg_pago_registro_auditoria
AFTER INSERT ON Pagos
FOR EACH ROW
BEGIN
    INSERT INTO Pagos_Auditoria (id_pago, id_cliente, id_membresia, fecha_pago, monto, metodo_pago, accion)
    VALUES (NEW.id_pago, NEW.id_cliente, NEW.id_membresia, NEW.fecha_pago, NEW.monto, NEW.metodo_pago, 'INSERT');
END$$
DELIMITER ;
```

4. Trigger: trg_evitar_reservas_clientes_sin_membresia

- **Descripción:** Impide que un cliente sin membresía activa realice una reserva.
- **Objetivo:** Garantizar que solo los clientes con membresías vigentes puedan asistir a clases.
- **Tablas involucradas:** Reservas, Membresias

```
DROP TRIGGER IF EXISTS trg_evitar_reservas_clientes_sin_membresia;
DELIMITER $$
CREATE TRIGGER trg_evitar_reservas_clientes_sin_membresia
BEFORE INSERT ON Reservas
FOR EACH ROW
BEGIN
    IF NOT EXISTS (
        SELECT 1
        FROM Membresias
        WHERE id_cliente = NEW.id_cliente
          AND estado = 'Activa'
```

```
        AND CURDATE() BETWEEN fecha_inicio AND fecha_fin
    ) THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'El cliente no tiene una membresía activa';
    END IF;
END$$
DELIMITER ;
```

5. Trigger: trg_clase_sin_empleado

- **Descripción:** Previene que se cree una clase sin asignar a un empleado.
- **Objetivo:** Asegura que toda clase tenga un instructor asignado.
- **Tablas involucradas:** Clases

```
DROP TRIGGER IF EXISTS trg_clase_sin_empleado;
DELIMITER $$
CREATE TRIGGER trg_clase_sin_empleado
BEFORE INSERT ON Clases
FOR EACH ROW
BEGIN
    IF NEW.id_empleado IS NULL THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'La clase debe tener un instructor asignado';
    END IF;
END$$
DELIMITER ;
```

Link del archivo SQL que contiene el script de los datos a insertar en la base de datos y el script de las vistas, funciones, procedimientos almacenados y triggers:

https://github.com/Miguesner/Proyecto-Centro-Deportivo

Formato:

Se añade a los archivos de Git Hub del proyecto Centro Deportivo, dos nuevos archivos SQL que corresponden a lo solicitado para esta entrega, uno para insertar datos en las tablas de la base de datos (Datos para la BD.sql) y el otro que contiene las vistas, funciones, procedimientos almacenados y triggers (Vistas, Funciones, Procedimientos Almacenados y Triggers.sql).