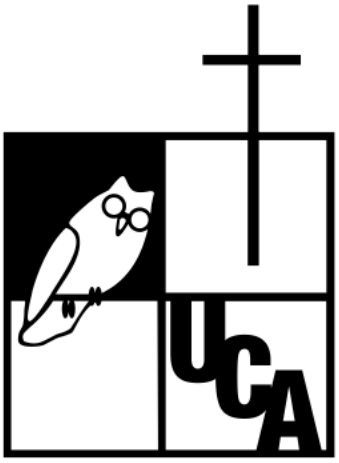


Facultad de
Ingeniería y Arquitectura

ADMINISTRACIÓN DE BASE DE DATOS

Informe para base de datos GimnasioBD

Integrantes:

Adriana Gabriela Campos Martínez #00302323

José Alfredo Hernández Soriano #00031323

Francisco José Cardona Mejía #00119123

Carlos Enrique Guevara Tobar #00039123

Néstor Alejandro Ayala Abarca #00133723

Catedrático:

James Edward Humberstone Morales

Descripción del sistema elegido y modelo de datos (ER)

El sistema elegido corresponde a un sistema de administración para un gimnasio cuyo objetivo es gestionar de manera eficiente la información relacionada con los socios, las membresías, los pagos, las clases, los entrenadores y las reservas.

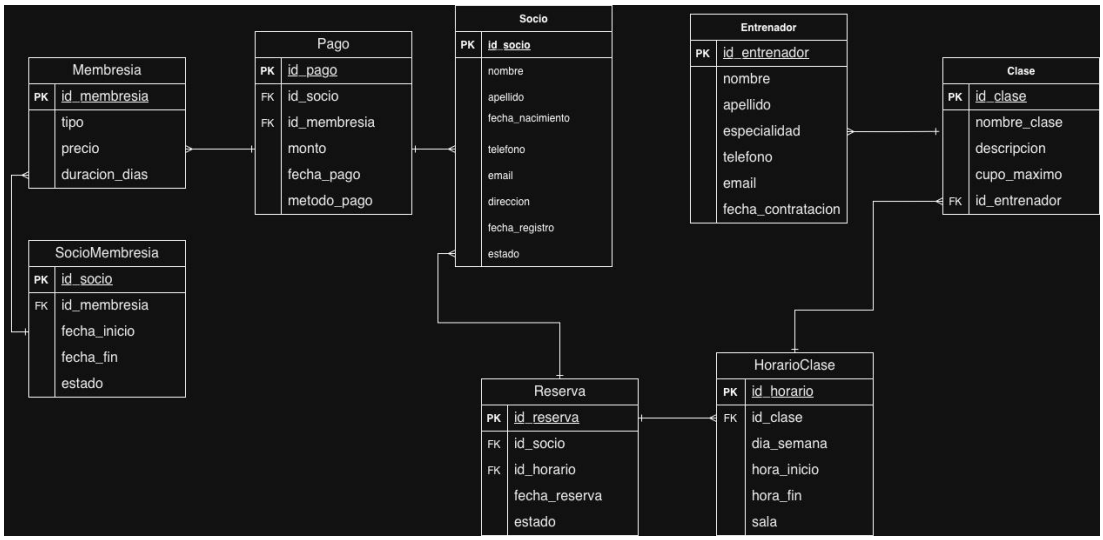
Este sistema permite llevar un control ordenado de los clientes del gimnasio, su estado de membresía, los pagos realizados, así como la programación de clases y la asignación de entrenadores. De esta forma se mejora la organización interna y se asegura un mejor control administrativo.

El modelo de datos se representa a través de un diagrama entidad relación que muestra la estructura lógica de la base de datos y define como se relacionan las distintas entidades entre sí. La entidad principal es Socio, ya que a partir de ella se gestionan las membresías, los pagos y las reservas.

El diagrama incluye entidades como "*Membresía"*, que define los planes disponibles, Pago, que registra las transacciones realizadas, Entrenador y Clase, que organizan las actividades del gimnasio, y *"Horario Clase"*, que establece los días y horas en que se imparten dichas clases. Además, se utiliza la entidad Socio Membresía para llevar el historial de membresías y la entidad Reserva para vincular a los socios con los horarios de clase.

Este modelo ER permite mantener la integridad de los datos y refleja de forma clara la relación entre los elementos principales del sistema, facilitando la administración y el control de las operaciones del gimnasio.

Modelo ER del GimnasioDB



Políticas de seguridad implementadas

Para mejorar la organización y la seguridad del sistema, se implementaron tres esquemas principales en la base de datos: **core**, **clases** y **admin**. Esta separación permite una mejor administración, control de permisos y una estructura más clara. A partir de esta organización, se propuso la creación de tres roles con diferentes niveles de acceso.

Explicación de los esquemas:

1. Core:

Contiene todas las tablas relacionadas a la información esencial del usuario y su relación con el gimnasio.
Incluye las tablas:

- **Membresia**
- **Socio**
- **SocioMembresia**
- **Pago**

Este esquema es fundamental, ya que gestiona la información crítica de membresías, pagos y registros del cliente.

2. Clases:

Contiene la información sobre las actividades internas del gimnasio relacionadas a la experiencia del usuario.
Incluye:

- **Entrenador**
- **Clase**
- **HorarioClase**
- **Reserva**

Estas tablas administran la interacción del usuario con los servicios del gimnasio: clases, horarios, reservas y entrenadores.

3. Admin:

Incluye exclusivamente las tablas destinadas al control administrativo del sistema, como auditorías internas.
Actualmente contiene:

- **Auditorías**, utilizadas para registrar eventos críticos como inserciones en tablas sensibles (por ejemplo, pagos).

Este esquema es clave para monitorear la integridad y las operaciones relevantes dentro de la base de datos.

Explicación de los Roles:

1. RolAdministrador

Es el rol con mayor nivel jerárquico.
Permisos:

- Acceso completo a todos los esquemas (incluyendo **dbo**).
- Permisos para crear procedimientos almacenados, funciones, vistas, triggers, auditorías y cualquier otro objeto.
- Lectura, inserción, actualización y eliminación en toda la base de datos.

Está destinado exclusivamente al personal encargado de la administración y monitoreo del sistema.

2. RolRecepción

Rol intermedio con permisos operativos, pensado para el personal administrativo del gimnasio.
Permisos:

- Acceso completo a los esquemas **core** y **clases**.
- Puede ejecutar procedimientos almacenados.
- Puede realizar **SELECT**, **INSERT** y **UPDATE**.
- *No tiene permiso para realizar DELETE*, como medida de seguridad para evitar pérdidas de información.

Este rol se encarga de la gestión diaria del gimnasio: socios, membresías, reservas, pagos y manejo de clases.

3. RolEntrenador

Rol con permisos de solo lectura para personal externo.
Permisos:

- Puede realizar únicamente **SELECT** sobre las tablas del esquema **clases**.
- No tiene permisos sobre el esquema core ni admin.

Este rol permite que los entrenadores consulten horarios, reservas y clases, sin comprometer información interna o sensible.

Imagen de Script utilizado:

```
--- Administrador acceso total ---
GRANT CONTROL ON SCHEMA::dbo TO RolAdministrador;
GRANT CONTROL ON SCHEMA::core TO RolAdministrador
GRANT CONTROL ON SCHEMA::clases TO RolAdministrador
GRANT CONTROL ON SCHEMA::admin TO RolAdministrador
GO

--Permisos adicionales
    GRANT CREATE FUNCTION TO RolAdministrador;
    GRANT CREATE VIEW TO RolAdministrador;
    GRANT CREATE TABLE TO RolAdministrador;
    GRANT CREATE PROCEDURE TO RolAdministrador;


-- Recepción acceso limitado
GRANT SELECT, INSERT, UPDATE ON SCHEMA::core TO RolRecepcion;
GRANT EXECUTE ON SCHEMA::core TO RolRecepcion
GRANT EXECUTE ON SCHEMA::clases TO RolRecepcion

DENY DELETE ON SCHEMA::core TO RolRecepcion;
GRANT SELECT ON SCHEMA::clases TO RolRecepcion;
GO


-- Entrenador acceso limitado
GRANT SELECT ON SCHEMA::clases TO RolEntrenador;
DENY SELECT ON SCHEMA::core TO RolEntrenador;
GO
```

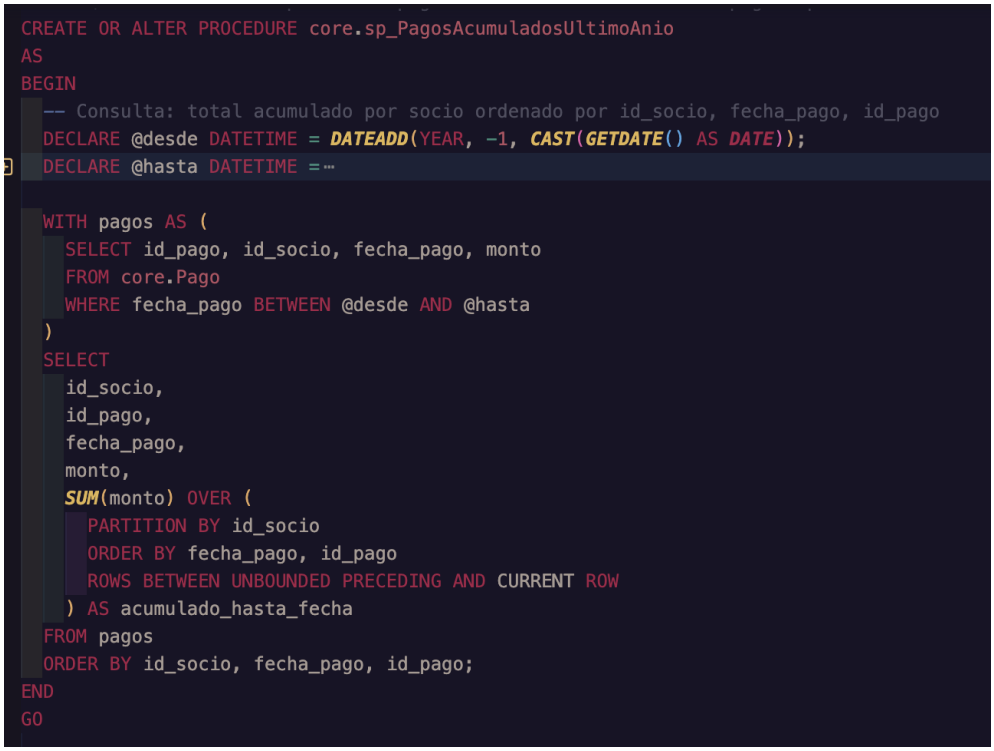
Evidencia de consultas optimizadas e índices aplicados.

Se realizaron 3 consultas usando índices para mejorar la eficiencia de las funciones ventanas:

- Función ventana sp_PagosAcumuladosUltimoAnio

Esta función muestra el historial de pagos de cada socio durante el último año, y calcula cuánto dinero ha ido pagando acumulado en el tiempo. Si un socio hace varios

pagos, el total se va acumulando en cada fila, dando seguimiento progresivo a sus pagos.



```sql
CREATE OR ALTER PROCEDURE core.sp_PagosAcumuladosUltimoAnio
AS
BEGIN
  -- Consulta: total acumulado por socio ordenado por id_socio, fecha_pago, id_pago
  DECLARE @desde DATETIME = DATEADD(YEAR, -1, CAST(GETDATE() AS DATE));
  DECLARE @hasta DATETIME =…

  WITH pagos AS (
    SELECT id_pago, id_socio, fecha_pago, monto
    FROM core.Pago
    WHERE fecha_pago BETWEEN @desde AND @hasta
  )
  SELECT
    id_socio,
    id_pago,
    fecha_pago,
    monto,
    SUM(monto) OVER (
      PARTITION BY id_socio
      ORDER BY fecha_pago, id_pago
      ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
    ) AS acumulado_hasta_fecha
  FROM pagos
  ORDER BY id_socio, fecha_pago, id_pago;
END
GO
```

- Función ventana sp_PagosMensuales

Esta consulta resume cuánto ingreso genera cada tipo de membresía por mes y luego calcula la media móvil de tres meses, lo cual permite ver tendencias y si las ventas están aumentando o disminuyendo en el tiempo.

```sql
CREATE OR ALTER PROCEDURE core.sp_PagosMensuales
AS
BEGIN
  WITH pagos_mensuales AS (
    SELECT
      id_membresia,
      YEAR(fecha_pago) AS año,
      MONTH(fecha_pago) AS mes,
      SUM(monto) AS total_mes,
      --fecha representativa del mes para ordenar
      DATEFROMPARTS(YEAR(fecha_pago), MONTH(fecha_pago), 1) AS primera_del_mes
    FROM core.Pago
    GROUP BY id_membresia, YEAR(fecha_pago), MONTH(fecha_pago), DATEFROMPARTS(YEAR(fecha_pago), MONTH(fecha_pago), 1)
  )
  SELECT
    id_membresia,
    año,
    mes,
    total_mes,
    ROUND(
      AVG(total_mes) OVER (
        PARTITION BY id_membresia
        ORDER BY primera_del_mes
        ROWS BETWEEN 2 PRECEDING AND CURRENT ROW
      ), 2) AS moving_avg_3m
  FROM pagos_mensuales
  ORDER BY id_membresia, año, mes;
END
GO
```

- Función ventana sp_MesesLlenos

Esta función compara cada mes con todos los demás meses en cuanto a número de reservas, asignando un ranking entre 0 y 1. Un valor de 1 significa que ese fue el mes con más reservas, y un valor de 0 es el mes menos activo.

```sql
CREATE OR ALTER PROCEDURE core.sp_MesesLlenos
AS
BEGIN
  --CONSLUTA
  WITH por_mes AS (
      SELECT
          DATEFROMPARTS(YEAR(fecha_reserva), MONTH(fecha_reserva), 1) AS mes,
          COUNT(*) AS total_reservas
      FROM clases.Reserva
      WHERE fecha_reserva >= DATEADD(YEAR, -1, CAST(GETDATE() AS DATE))  -- Ultimos 12 meses
      GROUP BY DATEFROMPARTS(YEAR(fecha_reserva), MONTH(fecha_reserva), 1)
  )
  SELECT
      mes,
      total_reservas,
      PERCENT_RANK() OVER (ORDER BY total_reservas) AS pct_rank_mes
  FROM por_mes
  ORDER BY total_reservas DESC;
END
GO
```

Prueba A:

Probando la función sp_PagosAcumuladosUltimoAnio sin índice creado

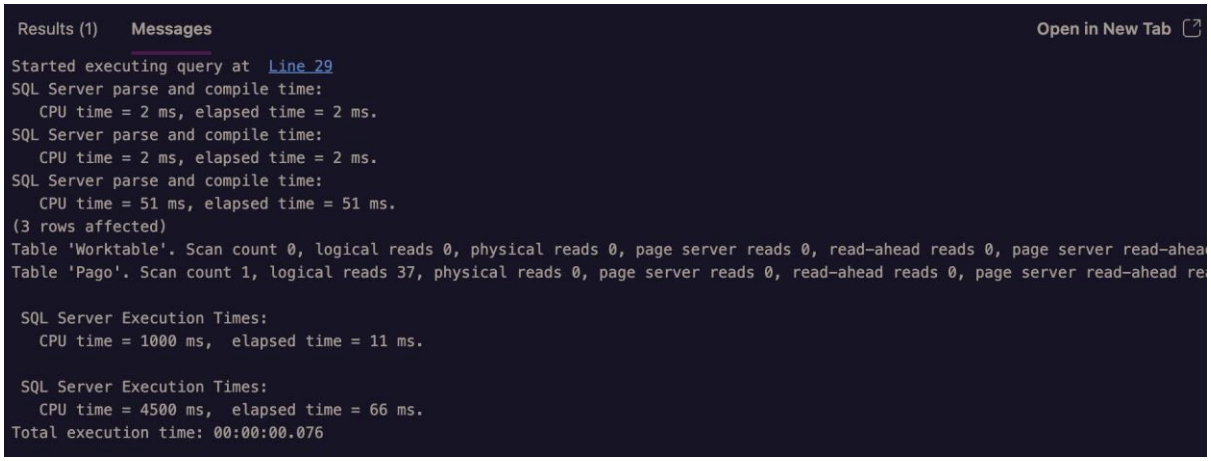
Probando la función sp_PagosAcumuladosUltimoAnio con índice creado

índice utilizado:

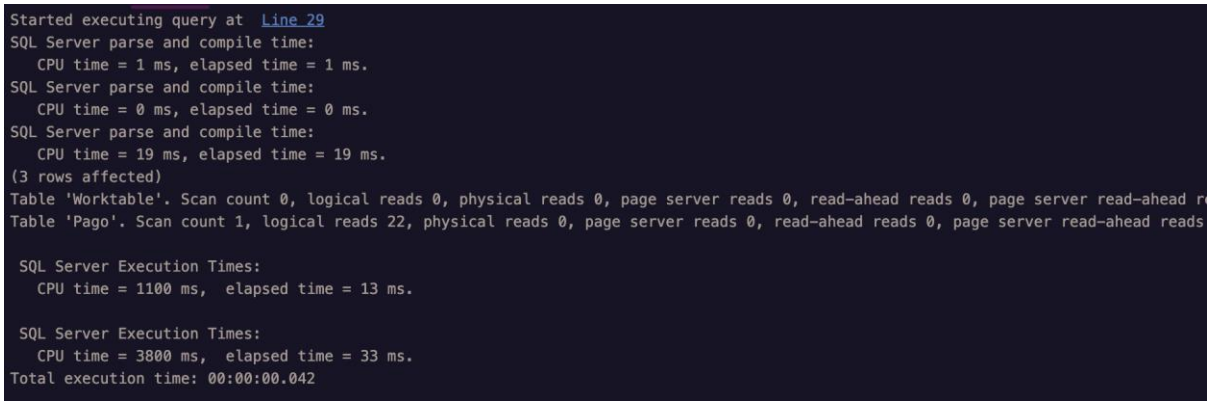
```
CREATE NONCLUSTERED INDEX IX_Pago_Socio_Fecha_IdPago_Cover
ON core.Pago (id_socio, fecha_pago, id_pago)
INCLUDE (monto, id_membresia, metodo_pago);
```

Prueba B:

Probando la función sp_PagosMensuales sin índice creado



Probando la función sp_PagosMensuales con índice creado

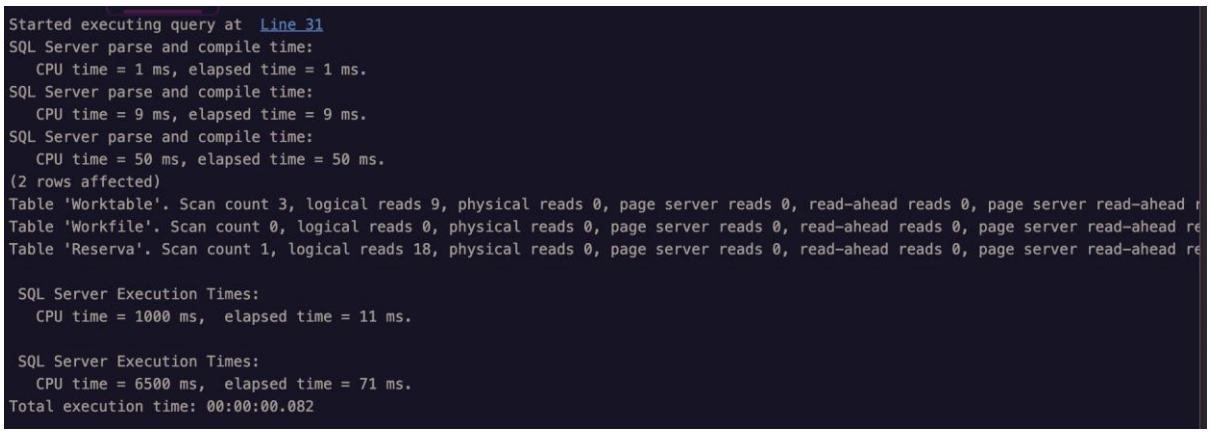


índice utilizado:

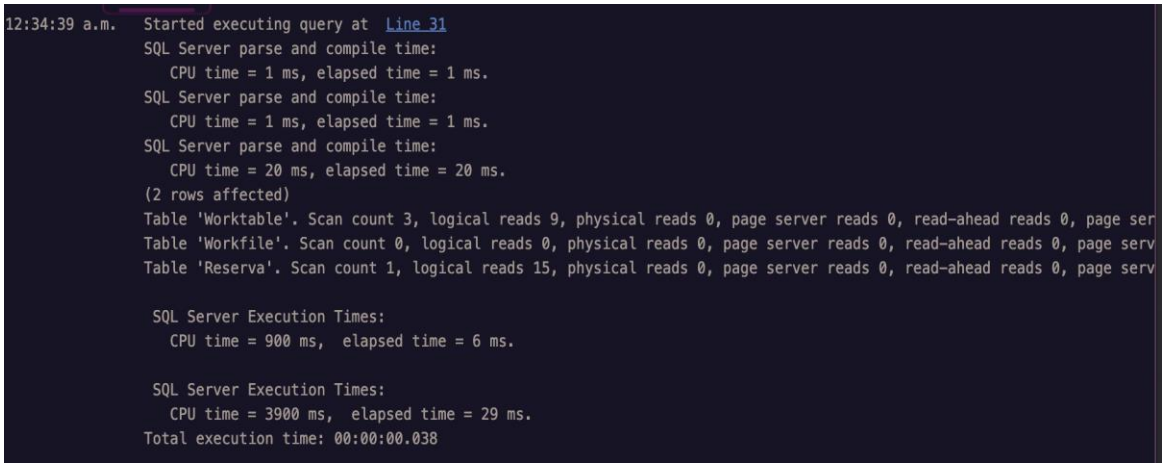
```
CREATE NONCLUSTERED INDEX IX_Pago_Membresia_Fecha
ON core.Pago (id_membresia, fecha_pago)
INCLUDE (monto);
```

Prueba C:

Probando la función sp_MesesLlenos sin índice creado



Probando la función sp_MesesLlenos con índice creado



índice utilizado:

```
CREATE NONCLUSTERED INDEX IX_Reserva_Fecha_Horario
ON clases.Reserva (fecha_reserva, id_horario)
INCLUDE (id_socio);
```

Estrategia de dimensionamiento, respaldo y recuperación

1. Plan de respaldo de la base de datos del gimnasio

Este es un pequeño resumen de cómo vamos a administrar la base de datos GimnasioDB, donde guardamos la información de socios, entrenadores, clases,

reservas y pagos. Como equipo no queremos depender de la suerte si un día falla el servidor, por eso decidimos diseñar nuestro propio plan de respaldo y recuperación.

Nuestro objetivo es que, si ocurre un problema (apagón, daño en el disco, errores humanos, etc.), podamos recuperar la base de datos sin perder demasiada información y en un tiempo razonable. Por eso definimos cada cuánto vamos a hacer copias, qué tipo de respaldos usaremos y cómo los restauraríamos paso a paso.

1.1 Agenda de respaldos que vamos a usar

Después de analizar el tamaño de nuestra base de datos y el uso que le daría un gimnasio real, como equipo acordamos trabajar con la siguiente agenda de respaldos:

Horario/Frecuencia	Tipo de backup	Qué se busca con esto
Domingo 00:00	FULL semanal	Guardamos una copia completa de GimnasioDB. Este será nuestro punto base de la semana.
Miércoles 00:00	DIFFERENTIAL intermedio	Solo copiamos los cambios desde el último FULL. Así aceleramos la restauración sin gastar tanto espacio.
Diariamente cada 15 minutos (06:00-22:00)	LOG de transacciones	transacciones recientes (pagos, reservas, altas de socios) para no perder casi nada si se daña el servidor.

Con este plan nosotros conseguimos al menos un respaldo completo cada semana, uno diferencial a mitad de semana y varios respaldos de LOG durante el día. De esta forma sentimos que los datos del gimnasio quedan bastante protegidos sin saturar el equipo con demasiados backups.

1.2 Estrategia de recuperación

Para tener claro el procedimiento, como equipo planteamos el siguiente escenario: el servidor del gimnasio se daña un miércoles a las 10:37 a.m. En ese momento ya tenemos:

- Un respaldo FULL del domingo
- Un respaldo DIFFERENTIAL del miércoles a las 00:00

- Varios respaldos de LOG generados cada 15 minutos (incluyendo el de las 10:30 a.m)

En ese caso nosotros seguiremos estos pasos para recuperar la base de datos:

1. Primero se buscará restaurar el último respaldo FULL del domingo en modo NORECOVERY

```
RESTORE DATABASE GimnasioDB
FROM DISK = 'D:\Backups\GimnasioDB\FULL\GimnasioDB_FULL_20251123.bak'
WITH NORECOVERY;
GO
```

2. Luego se aplica el respaldo DIFFERENTIAL del miércoles a las 00:00

```
RESTORE DATABASE GimnasioDB
FROM DISK = 'D:\Backups\GimnasioDB\DIFF\GimnasioDB_DIFF_20251126.bak'
WITH NORECOVERY;
GO
```

3. Se restauran los respaldos de LOG hasta las 10:30 a.m:

```
RESTORE LOG GimnasioDB
FROM DISK = 'D:\Backups\GimnasioDB\LOG\GimnasioDB_LOG_20251126_1030.trn'
WITH RECOVERY;
GO
```

Si quisiéramos dejar la base de datos justo antes de un error humano (por ejemplo, antes de borrar reservas por accidente), nosotros podríamos usar la opción STOPAT para indicar la hora exacta de corte:

```
RESTORE LOG GimnasioDB
FROM DISK = 'D:\Backups\GimnasioDB\LOG\GimnasioDB_LOG_20251126_1030.trn'
WITH STOPAT = '2025-11-26T10:25:00',
     RECOVERY;
GO
```

De esta manera se demuestra que la base de datos está preparada para una recuperación de datos.

1.3 Buenas prácticas a tomar en cuenta

- Aplicar la regla 3-2-1: mantener al menos tres copias de los datos (producción, copia local y copia externa o en la nube).
- Revisar de forma periódica que las carpetas de destino tengan espacio suficiente en disco.

- Nombrar los archivos de respaldo incluyendo el tipo y la fecha (por ejemplo: GimnasioDB_FULL_20251123.bak).
- Programar todos los respaldos como *Jobs del SQL Server Agent* para no depender de otras personas.
- Hacer restauraciones de prueba en una base de datos de laboratorio para confirmar que los .bak y .trn se puedan usar cuando sean necesarios.

Evidencia del dashboard en Power BI.

Nuestro dashboard se creó con datos comunes con la idea de demostrar la gran herramienta que es el *Power Bi*.

En el dashboard mostraremos la cantidad de socios que tiene el gimnasio, los ingresos registrados en el mes de apertura, los ingresos totales según la membresía y el total de pagos que se presentaron en efectivo.