**Universidad Católica Boliviana “San Pablo”**

**“Base de datos de Gimnasio”**



**Facultad:** Facultad de Ciencias Exactas

**Materia:** Base De Datos I

**Docente:** Marin Salazar Carmen Rosa

**Integrantes:**

* Leandro Joe Jaldin Gutierrez
* Camila Catorceno Orellana

1. **INTRODUCCION**

**Descripción del Problema**

El gimnasio Premier se enfrenta al reto de gestionar de manera eficiente y precisa la información relacionada con sus operaciones diarias. Con una creciente cantidad de clientes y una diversidad de servicios, es esencial contar con un sistema robusto de gestión de datos. El gimnasio ofrece cuatro servicios principales: pesas, zumba, cardio y box, cada uno con horarios específicos y entrenadores dedicados. Además, cada cliente paga una mensualidad fija de 250 Bs.

El sistema debe ser capaz de registrar y mantener la información de los clientes, incluyendo sus datos personales y número de teléfono. También debe administrar los servicios ofrecidos, detallando horarios y responsables. La gestión de inscripciones es crucial, permitiendo registrar el tiempo de suscripción de cada cliente, el costo y calculando automáticamente el total a pagar. Además, el sistema debe llevar un registro detallado de los pagos realizados por los clientes, controlar las asistencias a los diferentes servicios y gestionar la información de los entrenadores.

**Propuesta de Solución**

Nosotros proponemos una solución integral basada en una base de datos relacional que permitirá al Gimnasio Premier gestionar toda la información de manera eficiente y segura. La base de datos diseñada cumplirá con los siguientes objetivos:

**Gestión de Clientes:** Registrar y mantener la información de los clientes, garantizando la unicidad y evitando duplicaciones.

**Administración de Servicios:** Gestionar los detalles de los servicios ofrecidos, incluyendo horarios y entrenadores responsables.

**Registro de Inscripciones:** Permitir la inscripción de clientes, calculando automáticamente las fechas de inicio y fin, así como el total a pagar basado en la duración de la inscripción.

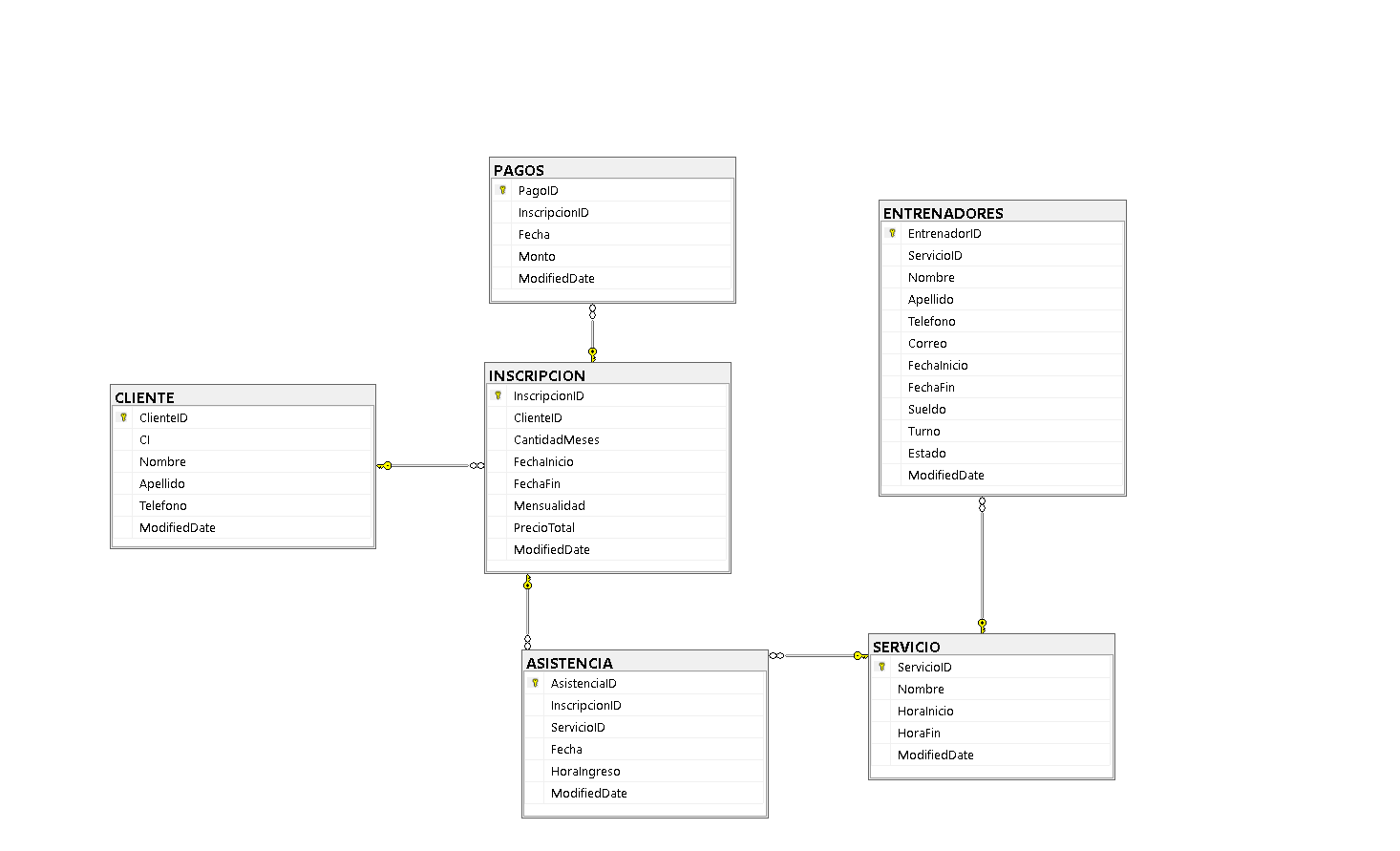
**Control de Pagos:** Mantener un registro detallado de los pagos realizados por los clientes, asegurando precisión en las transacciones.

**Seguimiento de Asistencias:** Rastrear las asistencias de los clientes a los diferentes servicios, asegurando que no se registren asistencias duplicadas.

**Gestión de Entrenadores:** Administrar la información de los entrenadores, incluyendo sus datos personales, servicios impartidos y estado de empleo.

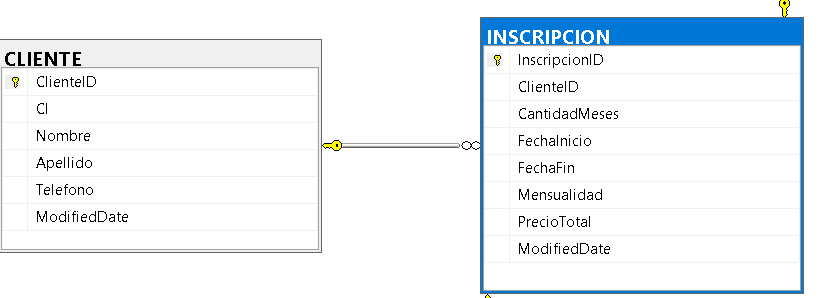
Esta solución proporcionará al gimnasio las herramientas necesarias para mejorar la eficiencia operativa, reducir errores y ofrecer un mejor servicio a sus clientes.

1. **Diseño**



**Entidades relaciones y atributos**

**1.Tabla CLIENTE**

****

**ClienteID : Identificador único de cada cliente**

**Ci: Es el carnet de identidad del cliente**

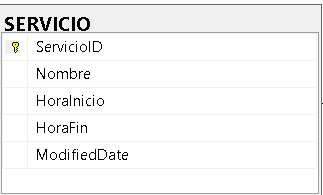
**Nombre: Nombre del cliente**

**Apellido: Apellido del cliente**

**Telefono: Numero de teléfono del cliente, este debe ser único**

**ModifieDate: Fecha de la ultima modificación del registro**

1. **Tabla SERVICIO**

****

**ServicioID: Identificador único de cada servicio**

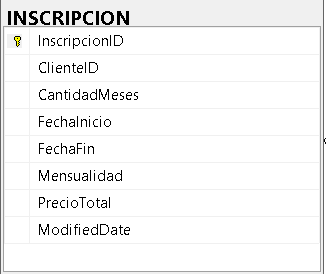
**Nombre: Nombre del servicio**

**HoraInicio: Hora que inicia el servicio**

**HoraFin: Hora que finaliza el servicio**

**ModifeDate: Fecha de la última modificación del registro**

1. **Tabla INSCRIPCION**



**InscripcionID: Identificador único de la inscripción**

**ClienteID: Referencia al cliente que realiza la inscripción**

**CantidadMeses: Numero de meses de la inscripción (por cuantos meses entrenara)**

**FechaInicio: Fecha de inicio de la inscripción**

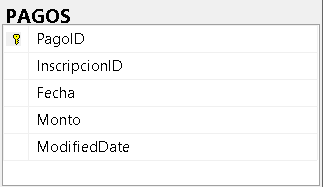
**FechaFin: Fecha de finalización de la inscripción**

**Mensualidad: Monto mensual de la inscripción**

**PrecioTotal: Precio total de la inscripción**

**ModifieDate: fecha de la última modificación del registro**

1. **Tabla PAGOS**

****

**PagoID: Identificador Unico de cada Pago**

**InscripcionID: Referencia al cliente que realiza la inscripción**

**Fecha: Fecha de pago**

**Monto: Monto pagado**

**ModifeDate: Fecha de ultima modificación del registro**

1. **Tabla Asistencia**

****

**AsistenciaID: Identificador único de cada asistencia**

**Inscripcion: Referencia a la inscripción del cliente**

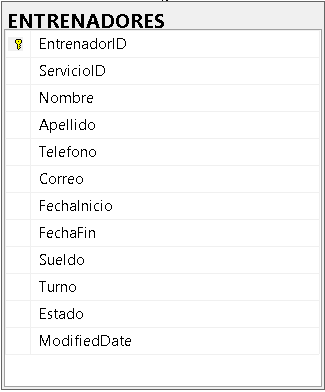
**ServicioID: Referencia al servicio que asiste al cliente**

**Fecha: Fecha de asistencia**

**HoraIngreso: Hora de ingreso del cliente**

**ModifieDate: Fecha de la ultima modificación del registro**

1. **Tabla ENTRENADORES**



**EntrenadorID: Identificador único de cada entrenador.**

**ServicioID: Referencia al servicio que imparte el entrenador.**

**Nombre : Nombre del entrenador.**

**Apellido: Apellido del entrenador.**

**Telefono: Número de teléfono del entrenador.**

**Correo: Correo electrónico del entrenador.**

**FechaInicio: Fecha de inicio del contrato del entrenador.**

**FechaFin: Fecha de finalización del contrato del entrenador, si aplica.**

**Sueldo: Sueldo del entrenador.**

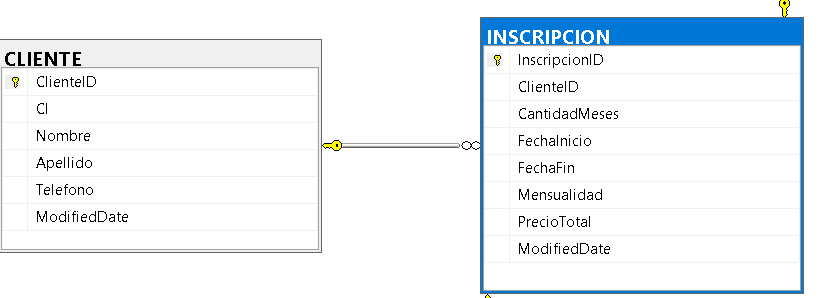
**Turno: Turno del entrenador (mañana, tarde, noche).**

**Estado: Estado del entrenador (Activo o Despedido), calculado basado en la fecha de finalización.**

**ModifiedDate: Fecha de la última modificación del registro.**

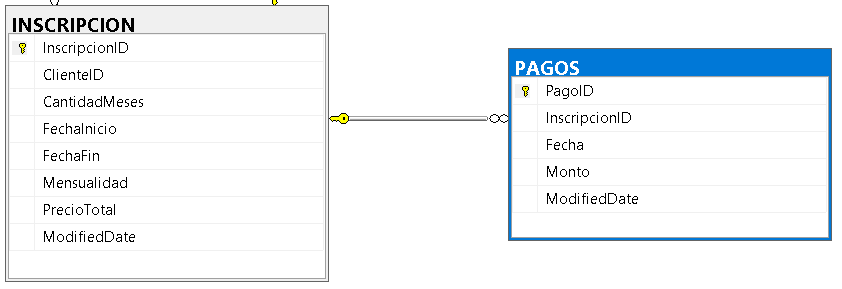
**Relaciones entre las entidades**

1. **Cliente a Inscripciones**

****

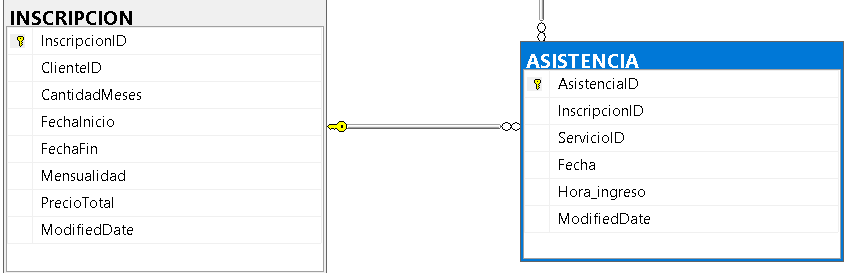
**Un cliente puede tener múltiples inscripciones**

1. **Inscripción a pagos**



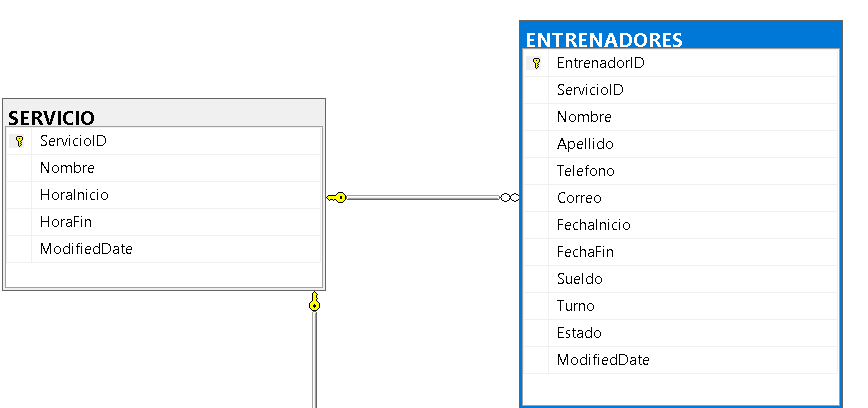
**Una inscripción puede tener múltiples pagos**

1. **Inscripción a Asistencia**



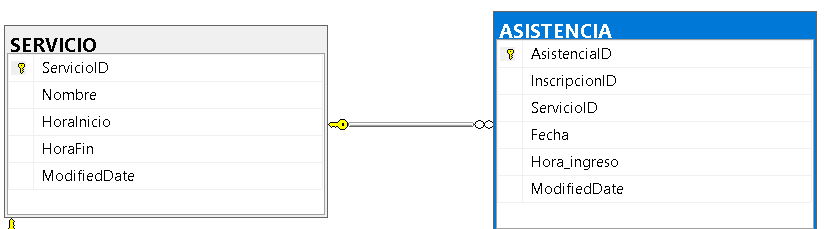
**Una inscripción puede tener múltiples asistencias**

1. **Servicio a Entrenadores**



**Un servicio puede ser impartido a múltiples entrenadores**

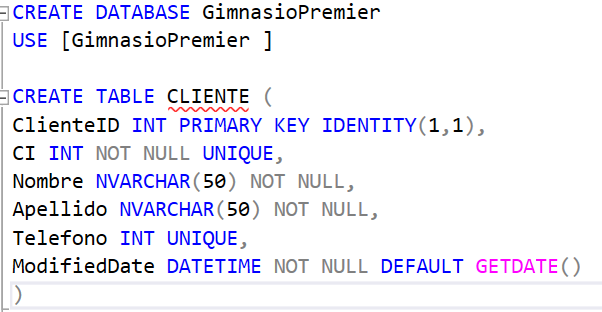
1. **Servicio a Asistencia**



**Un servicio puede tener múltiples asistencias registradas**

1. Scripts

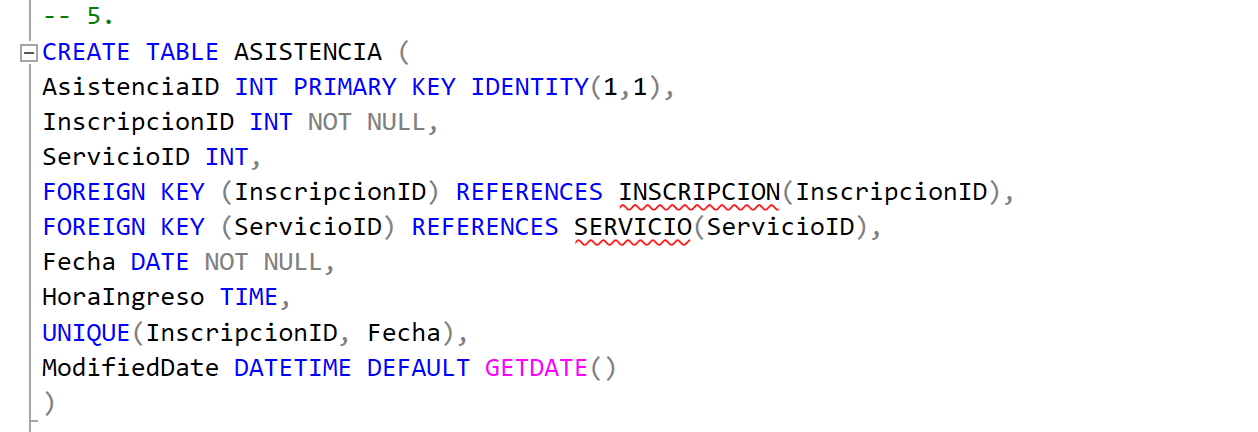
DDL:



Interfaz de usuario gráfica, Texto

Descripción generada automáticamente con confianza mediaInterfaz de usuario gráfica, Texto, Aplicación

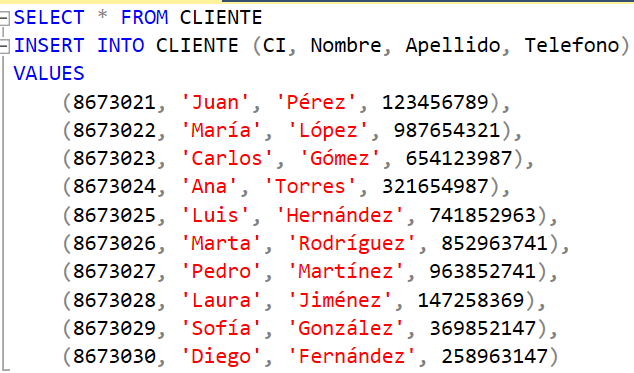
Descripción generada automáticamenteInterfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamenteInterfaz de usuario gráfica, Aplicación

Descripción generada automáticamenteInterfaz de usuario gráfica

Descripción generada automáticamente

DML:

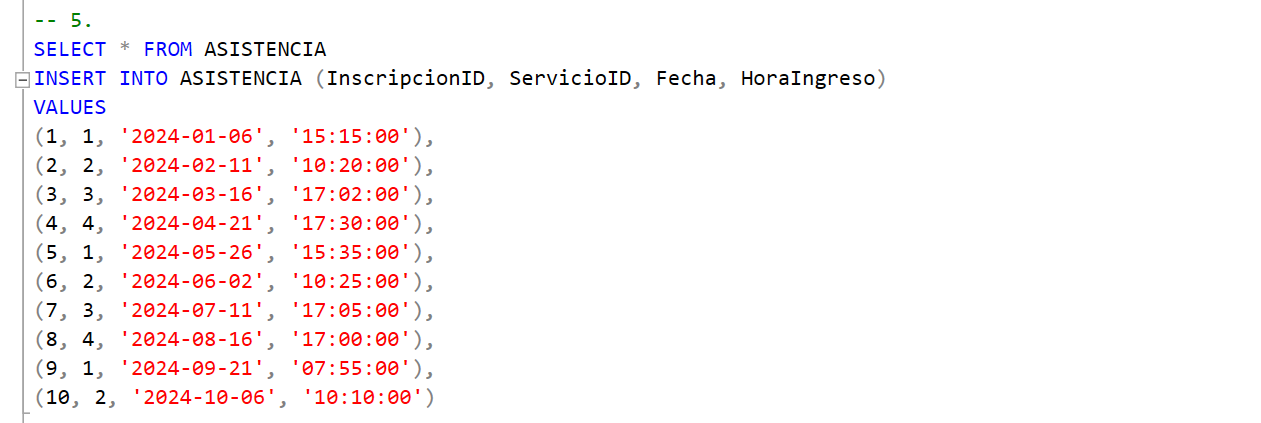


Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

Texto

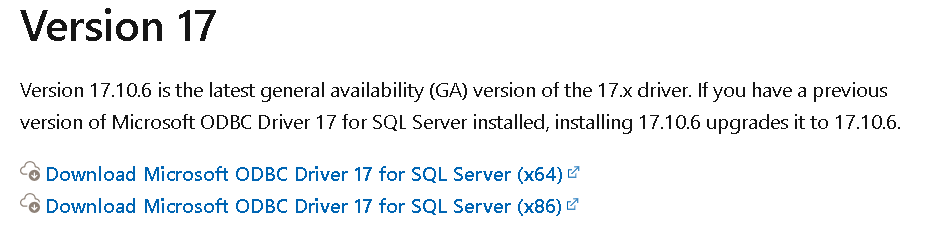
Descripción generada automáticamenteTexto

Descripción generada automáticamente con confianza mediaTexto

Descripción generada automáticamente

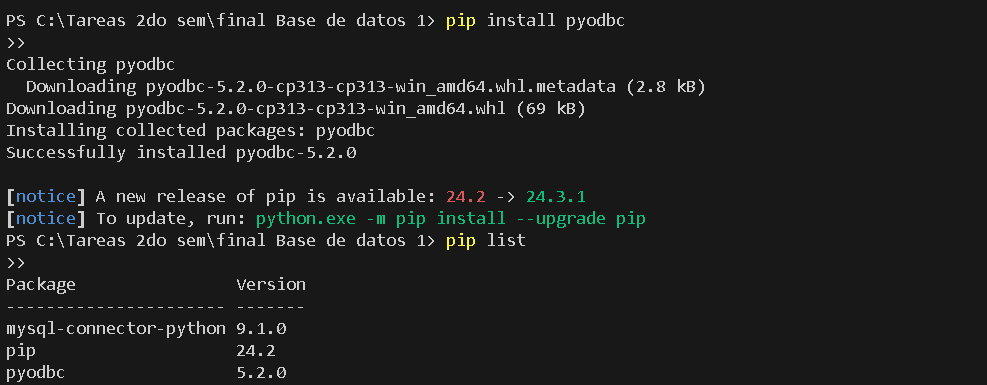
**Migración de Datos**

1. **Primero instalamos**

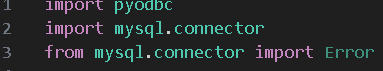


* **La instalación fue de (x64) por la información del sistema**
* **Se utiliza para conectarse a bases de datos MySQL.**

1. **Instalación de la librería Pyodbc (open database connectivity)**

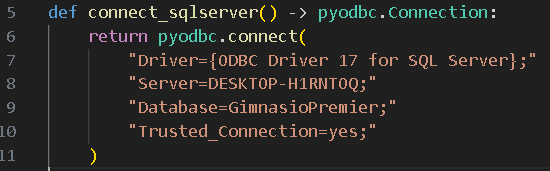


1. **Código Migración**
2. **Conexión a SQL SERVER**



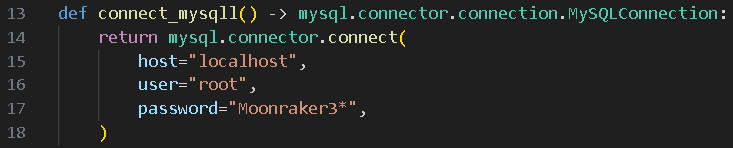
* **pyodbc para SQL Server**
* **mysql.connector para MySQL**

1. **conceccion sql server**



* **pyodbc: Conecta a SQL Server.**
* **Driver: Controlador para SQL Server.**
* **Server: Nombre del servidor (ej. DESKTOP-H1RNTOQ).**
* **Database: Nombre de la base de datos (ej. GimnasioPremier).**
* **Trusted\_Connection: Usa autenticación de Windows.**
* **cursor: Ejecuta consultas como SELECT o INSERT.**

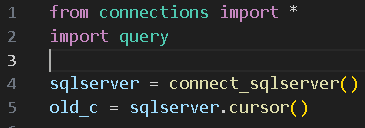
1. **Conexión a MySQL**



* **def connect\_mysqll(): Declara una función llamada connect\_mysqll que se utiliza para establecer una conexión con una base de datos MySQL.mysql.connector.connection.MySQLConnection: Especifica el tipo de valor que devuelve la función: un objeto MySQLConnection, que representa la conexión con la base de datos MySQL.**
* **mysql.connector.connect: Llama a la función para conectarse al servidor MySQL.**
* **host="localhost": Indica que el servidor MySQL está en la misma máquina donde se ejecuta el código.**
* **user="root": Define el usuario que se utilizará para conectarse a MySQL (en este caso, el usuario por defecto root).**

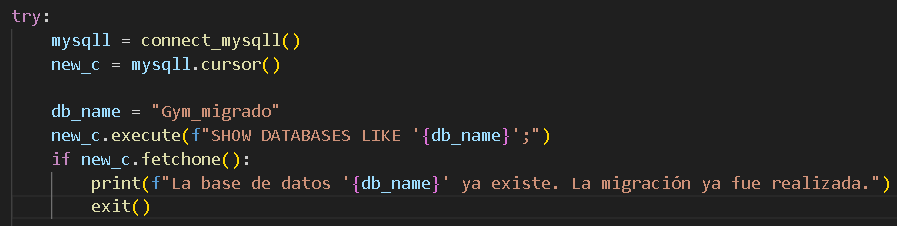
**Archivo main 3**

1. **librerías**



* **from connections import \*: Importa todas las funciones y variables definidas en el archivo connections.py. Esto incluye las funciones connect\_sqlserver y connect\_mysqll, que permiten conectar a las bases de datos SQL Server y MySQL respectivamente.**
* **import query: Importa el archivo o módulo query. Este podría contener funciones o sentencias SQL adicionales que se usan para ejecutar consultas o modificar las bases de datos.**
* **sqlserver = connect\_sqlserver(): Llama a la función connect\_sqlserver del archivo connections.py para establecer una conexión con la base de datos SQL Server y almacena el objeto de conexión en la variable sqlserver.**
* **old\_c = sqlserver.cursor(): Crea un cursor llamado old\_c a partir de la conexión sqlserver. Este cursor se usa para ejecutar consultas SQL (como SELECT, INSERT, etc.) en la base de datos SQL Server.**

1. **Conexión a MySQL**



* **try:: Inicia un bloque de código para manejar excepciones. Si ocurre algún error durante la ejecución del código dentro del bloque try, será capturado por el bloque except (si se incluye). Esto ayuda a evitar que el programa falle de manera abrupta.**
* **mysqll = connect\_mysqll(): Llama a la función connect\_mysqll (definida previamente) para establecer una conexión con la base de datos MySQL. El objeto de conexión se almacena en la variable mysqll.**
* **new\_c = mysqll.cursor(): Crea un cursor (new\_c) a partir de la conexión mysqll. Este cursor es necesario para ejecutar consultas SQL en la base de datos MySQL.**
* **db\_name = "Gym\_migrado": Define una variable llamada db\_name que contiene el nombre de la base de datos que se verificará.**
* **new\_c.execute(f"SHOW DATABASES LIKE {db\_name};"): Ejecuta una consulta SQL en MySQL para comprobar si existe una base de datos con el nombre almacenado en db\_name (Gym\_migrado).**
* **SHOW DATABASES LIKE {db\_name};: Es una consulta MySQL que lista las bases de datos cuyo nombre coincida con Gym\_migrado.**
* **if new\_c.fetchone():: Verifica si la consulta anterior devolvió algún resultado.**
* **new\_c.fetchone(): Obtiene la primera fila del resultado de la consulta. Si no hay resultados (es decir, la base de datos no existe), devuelve None.**
* **print(f"La base de datos '{db\_name}' ya existe. La migración ya fue realizada."):**
* **Imprime un mensaje indicando que la base de datos Gym\_migrado ya existe, por lo que no es necesario realizar la migración nuevamente.**
* **exit():**

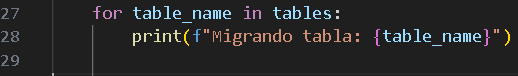
**Finaliza la ejecución del programa inmediatamente si la base de datos ya existe.**

1. **Tablas a migrar**



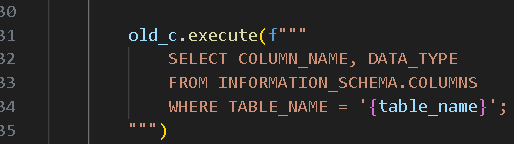
* **Estas son las tablas que se migraran de Sql server a MySQL**
* **Esas tablas actualmente están en sql server**

1. **Iteracion sobre las tablas**



* **Lo que hace el for es iterar sobre cada tabla declaramos nuestra variable table\_name y recorrerá cada tabla de la variable tabla**
* **La variable tables es una lista dentro de ella se encuentra nuestras tablas de sql**

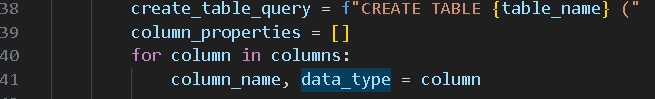
1. **Obtenemos el esquema de nuestras tablas**



* **old\_c.execute(...):**
* **Utiliza el cursor old\_c (previamente creado) para ejecutar una consulta SQL en la base de datos SQL Server.**
* **Consulta SQL:**
* **SELECT COLUMN\_NAME, DATA\_TYPE:**
* **Recupera los nombres de las columnas (COLUMN\_NAME) y los tipos de datos (DATA\_TYPE) de una tabla específica.**
* **FROM INFORMATION\_SCHEMA.COLUMNS:**
* **La consulta se dirige a la vista INFORMATION\_SCHEMA.COLUMNS, que contiene metadatos sobre las columnas de todas las tablas de la base de datos.**
* **WHERE TABLE\_NAME = {table\_name}:**
* **Filtra los resultados para devolver solo las columnas de la tabla cuyo nombre es igual al valor de table\_name.**
* **table\_name es una variable que contiene el nombre de la tabla de la que deseas obtener la información.**

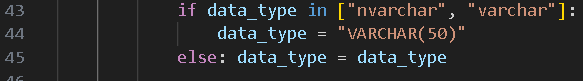
**Propósito: Este bloque de código recupera los nombres de las columnas y sus tipos de datos de una tabla específica en SQL Server. Esta información se utiliza, por ejemplo, para recrear la misma tabla en MySQL durante el proceso de migración.**

1. **Creación de tablas en MySQL**



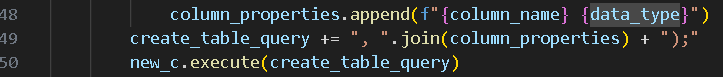
* **create\_table\_query = f"CREATE TABLE {table\_name} (":**
* **Inicializa una cadena con la estructura base de la sentencia CREATE TABLE, donde {table\_name} es el nombre dinámico de la tabla.**
* **column\_properties = []:**
* **Crea una lista vacía para almacenar los atributos de cada columna (nombre y tipo de dato) que se agregarán a la sentencia CREATE TABLE.**
* **for column in columns::**
* **Itera sobre cada columna en la lista columns, donde cada elemento es una tupla con información de la columna.**
* **column\_name, data\_type = column:**
* **Extrae los valores individuales de la tupla:**
* **column\_name: El nombre de la columna (por ejemplo, 'id' o 'nombre').**
* **data\_type: El tipo de dato de la columna en SQL Server (por ejemplo, 'int' o 'nvarchar')**

1. **Mapear los tipos de datos**



* **if data\_type in ["nvarchar", "varchar"]:**
* **Comprueba si el tipo de dato (data\_type) pertenece a la lista ["nvarchar", "varchar"].**
* **data\_type = "VARCHAR(50)"**
* **Si la condición se cumple, el tipo de dato se mapea a VARCHAR(50) para MySQL.**
* **else: data\_type = data\_type**
* **Si el tipo de dato no es nvarchar o varchar, mantiene su valor original sin cambios.**

1. **Añadimos el tipo de dato y el nombre de la columna**



**column\_properties.append(f"{column\_name} {data\_type}"):**

* **Agrega a la lista column\_properties una definición de columna con su nombre y tipo de dato formateado.**
* **{column\_name}: Inserta el nombre de la columna.**
* **{data\_type}: Inserta el tipo de dato, previamente mapeado al formato compatible con MySQL.**
* **Ejemplo: Si column\_name = "nombre" y data\_type = "VARCHAR(50)", el resultado será:**

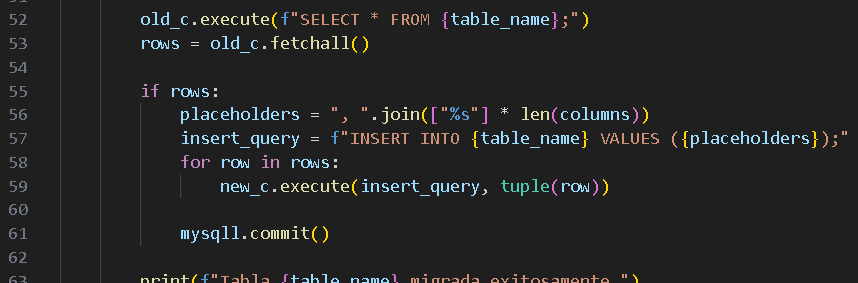
**sql**

**Copy code**

**nombre VARCHAR(50)**

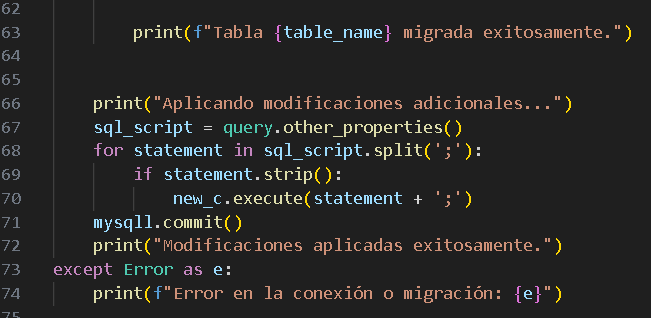
* **create\_table\_query += ", ".join(column\_properties) + ");":**
* **Une todas las definiciones de columna contenidas en column\_properties con una coma y las agrega al final de la sentencia SQL CREATE TABLE.**
* **.join(column\_properties): Combina todos los elementos de la lista column\_properties en una sola cadena separada por comas.**
* **+ ");": Cierra la sentencia CREATE TABLE con el paréntesis final y el punto y coma.**
* **Ejemplo completo: Si column\_properties = ['id INT', 'nombre VARCHAR(50)'], la sentencia resultante será:**
* **sql**
* **Copy code**
* **CREATE TABLE CLIENTE (id INT, nombre VARCHAR(50));**

1. **Sentencia sql**



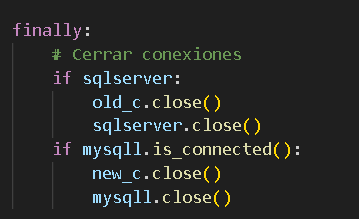
* **old\_c.execute(f"SELECT \* FROM {table\_name};"): Ejecuta una consulta en SQL Server para obtener todos los registros de la tabla.**
* **rows = old\_c.fetchall(): Recupera los registros y los almacena como una lista de tuplas en rows.**
* **if rows:: Verifica si hay registros en la tabla.**
* **placeholders = ", ".join(["%s"] \* len(columns)): Crea una lista de marcadores (%s) separados por comas según el número de columnas.**
* **insert\_query = f"INSERT INTO {table\_name} VALUES ({placeholders});": Prepara una consulta de inserción para MySQL.**
* **for row in rows:: Itera por cada registro obtenido.**
* **new\_c.execute(insert\_query, tuple(row)): Inserta cada registro en la tabla de MySQL.**
* **mysqll.commit(): Guarda los cambios en la base de datos MySQL.**
* **print(f"Tabla {table\_name} migrada exitosamente."): Muestra un mensaje indicando que la tabla fue migrada.**

1. **Prints**



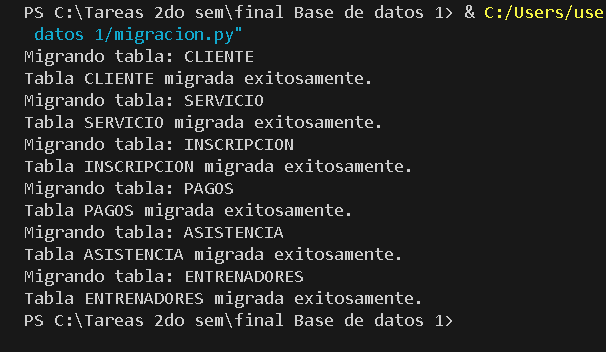
* **print(f"Tabla {table\_name} migrada exitosamente."): Muestra un mensaje indicando que la tabla ha sido migrada correctamente.**
* **print("Aplicando modificaciones adicionales..."): Indica que se están aplicando cambios adicionales a la base de datos.**
* **sql\_script = query.other\_properties(): Obtiene un script SQL desde el módulo query, que contiene propiedades adicionales para las tablas.**
* **for statement in sql\_script.split(';'):: Divide el script SQL en sentencias individuales basándose en los puntos y comas (;).**
* **if statement.strip():: Verifica que la sentencia no esté vacía (evita errores).**
* **new\_c.execute(statement + ';'): Ejecuta cada sentencia SQL para aplicar las modificaciones.**
* **mysqll.commit(): Confirma los cambios realizados en la base de datos MySQL.**
* **print("Modificaciones aplicadas exitosamente."): Indica que las modificaciones adicionales se aplicaron correctamente.**
* **except Error as e:: Captura cualquier error que ocurra durante la conexión o migración.**
* **print(f"Error en la conexión o migración: {e}"): Muestra el mensaje de error para identificar el problema.**

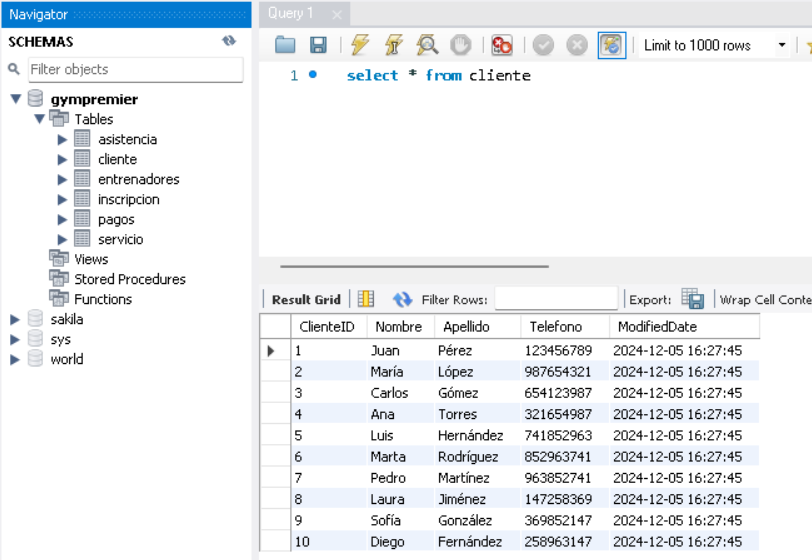
1. **Cierre de concexiones**



* **finally::**
* **Indica un bloque que se ejecutará siempre, independientemente de si hubo o no un error en el programa. Se utiliza aquí para cerrar las conexiones.**
* **if sqlserver::**
* **Verifica si la conexión a SQL Server (sqlserver) está activa.**
* **old\_c.close(): Cierra el cursor asociado a SQL Server.**
* **sqlserver.close(): Cierra la conexión con la base de datos SQL Server.**
* **if mysqll.is\_connected()::**
* **Comprueba si la conexión a MySQL (mysqll) sigue activa.**
* **new\_c.close(): Cierra el cursor asociado a MySQL.**
* **mysqll.close(): Cierra la conexión con la base de datos MySQL.**

1. **Salida del código de migración**



1. **Confirmación de migración de SQL Server a MySQL**