**1. Interfaz entre Python y SQL Server:**

Debido a que tienes la validación de datos en Python y la base de datos en SQL Server, necesitarás establecer una conexión entre estos dos. **Python** es excelente para realizar operaciones de validación y procesamiento, mientras que **SQL Server** te ayudará con el almacenamiento y manejo de datos a gran escala.

**Librería recomendada:**

Puedes usar **pyodbc** o **sqlalchemy** para establecer la conexión desde Python hacia SQL Server. **pyodbc** es más ligero y directo, mientras que **sqlalchemy** puede proporcionar un enfoque más estructurado (y a menudo mejor para proyectos más grandes), ya que es un ORM.

**2. Proceso de conexión:**

Te recomiendo un flujo de trabajo que unifique todo, donde Python maneje tanto la validación de datos como la inserción de los datos validados a SQL Server.

**a. Conexión a SQL Server desde Python:**

Aquí tienes un ejemplo usando pyodbc para conectarte a SQL Server:

python

Copiar código

import pyodbc

# Conexión a la base de datos

connection = pyodbc.connect('DRIVER={ODBC Driver 17 for SQL Server};'

'SERVER=Nombre\_Servidor;'

'DATABASE=Prestamo\_universal;'

'UID=usuario;'

'PWD=contraseña;')

cursor = connection.cursor()

Si prefieres usar sqlalchemy, el enfoque sería algo así:

python

Copiar código

from sqlalchemy import create\_engine

# Crear el motor de conexión

engine = create\_engine("mssql+pyodbc://usuario:contraseña@Nombre\_Servidor/Prestamo\_universal?driver=ODBC+Driver+17+for+SQL+Server")

connection = engine.connect()

**3. Validación e Inserción de Datos:**

Tu script de validación en Python debería funcionar de manera similar a como ya lo tienes, con el paso final de insertar los datos validados en SQL Server.

**Flujo sugerido:**

1. **Lectura y Validación de CSV:** Continuar con el proceso de validación que ya tienes.
2. **Inserción de Datos Validados:** Una vez que los datos están validados, puedes usar Python para ejecutar BULK INSERT o directamente insertar filas una por una mediante SQL queries.

**Ejemplo de inserción usando pyodbc:**

python

Copiar código

# Después de la validación de datos

sql\_query = """

INSERT INTO MD\_clientes (ti\_documento, nu\_documento, nombre\_cliente, email\_cliente, ...)

VALUES (?, ?, ?, ?, ...)

"""

# Ejemplo con datos validados

data = ('DNI', '12345678', 'Nombre Cliente', 'correo@ejemplo.com', ...)

cursor.execute(sql\_query, data)

connection.commit() # Confirma la transacción

Puedes iterar sobre los registros validados y ejecutar múltiples inserciones de esta forma. Para una mayor eficiencia, podrías insertar bloques de datos en lotes (batch inserts).

**4. Generación de Cronogramas desde Datos de SQL Server:**

Ahora que los datos están en SQL Server, tu script para generar cronogramas puede obtener directamente los datos de la base de datos en lugar de introducirlos manualmente.

**Ejemplo de cómo recuperar datos:**

python

Copiar código

sql\_query = """

SELECT monto\_prestamo, tasa\_interes, num\_cuotas, fecha\_inicio\_prestamo

FROM MD\_prestamos

WHERE id\_cliente = ?

"""

id\_cliente = 1 # Ejemplo

cursor.execute(sql\_query, id\_cliente)

result = cursor.fetchall()

# Generar cronograma con los resultados

for row in result:

monto\_prestamo, tasa\_interes, num\_cuotas, fecha\_inicio\_prestamo = row

# Aquí puedes llamar a tu función que genera el cronograma con estos valores

**5. Automatización del flujo:**

Si quieres automatizarlo, te recomiendo un **script Python maestro** que siga este flujo:

1. **Cargar CSV** -> **Validar Datos**.
2. **Insertar datos validados** en SQL Server (MD\_clientes y MD\_inversionistas).
3. **Obtener datos** para generar cronogramas.
4. **Generar cronograma** usando los datos obtenidos.
5. (Opcional) **Exportar el cronograma** en Excel/PDF o enviar por correo con SendGrid.

De esta forma, todo el flujo de carga y generación de cronogramas está integrado.

**6. Errores y transacciones:**

Al manejar la inserción de datos, siempre es una buena práctica gestionar errores y asegurarte de que las transacciones se completen correctamente. Por ejemplo, puedes agregar manejo de excepciones en Python:

python

Copiar código

try:

cursor.execute(sql\_query, data)

connection.commit()

except pyodbc.Error as e:

print(f"Error: {e}")

connection.rollback() # En caso de error, revertir la transacción

**7. Mejora futura: API RESTful o Pipeline ETL:**

Una vez que todo esté integrado, un siguiente paso de evolución podría ser crear un **API RESTful** para exponer estos servicios (validación, carga y generación de cronogramas) a través de endpoints. Esto sería útil si quieres que otros sistemas interactúen con tu base de datos de forma más flexible.

Otro enfoque sería crear un **pipeline de ETL** (Extracción, Transformación y Carga) más robusto para manejar la extracción de datos de fuentes externas, la validación automática, y la carga a SQL Server.

Con esta estructura, puedes tener un flujo de trabajo sólido, escalable y eficiente que te permitirá trabajar con facilidad tanto con los datos como con la generación de cronogramas.