Sistemas de Gestión de Bases de Datos con Python
ONLINE 2025 I LIMA

Tarea Sesión N° 02

Situación Actual

En entornos empresariales, los datos almacenados en hojas de cálculo como Google Sheets son omnipresentes debido a su facilidad de uso y colaboración en tiempo real. Sin embargo, para análisis avanzados, escalabilidad y eficiencia, es esencial migrar estos datos a bases de datos relacionales. Esta migración no solo optimiza el rendimiento de las consultas, sino que también permite una gestión estructurada y automatizada de los datos.

En esta tarea, nos enfocaremos en potenciar el uso de `SQLAlchemy` como ORM (Object-Relational Mapping) para abstraer la complejidad de las interacciones con la base de datos, facilitando un desarrollo más intuitivo y mantenible en Python.

Objetivos

1. Integración de Google Sheets a SQLite usando Python: Extraer datos de hojas de cálculo en la nube y cargarlos en una base de datos local relacional, implementando un flujo ETL (Extracción, Transformación, Carga) básico.

2. Creación y Gestión de Esquemas de Base de Datos con SQLAlchemy: Utilizar el enfoque declarativo de SQLAlchemy para definir modelos de datos, generar tablas automáticamente y manejar relaciones complejas.

3. Ejecución Avanzada de Consultas y Operaciones ORM: Realizar consultas SQL a través del ORM de SQLAlchemy, incluyendo joins, filtros, agregaciones y actualizaciones.

La tarea comprende los siguientes puntos

- Conexión de Python con Fuentes de Datos en la Nube: Uso de `gspread` para autenticación con OAuth 2.0 y extracción de datos de Google Sheets. Integración con `pandas` para transformaciones iniciales.
- Implementación de ETL básico con SQLAlchemy como núcleo:
  - **Extracción**: lectura de datos y conversión a DataFrame.
  - **Transformación**: operaciones con pandas (filtrado, agregación, joins).
  - **Carga**: definición de modelos en SQLAlchemy con anotaciones de tipo y creación de la base de datos.
- Ejecución de consultas SQL mediante ORM.
- Comparación de rendimiento ORM vs SQL crudo.

Guía referencial del proceso a implementar

Datos de ejemplo (Simulación de Google Sheets)

Columnas: ID, Producto, Cantidad, Precio, Fecha, Cliente_ID. Filas: 5 entradas de ejemplo.

Listing 1: Conexión a Google Sheets con gspread

```
import gspread
from gspread_dataframe import get_as_dataframe
import pandas as pd
from oauth2client.service_account import ServiceAccountCredentials

# Configura credenciales
scope = ['https://spreadsheets.google.com/feeds',
         'https://www.googleapis.com/auth/drive']

creds = ServiceAccountCredentials.from_json_keyfile_name('credentials.json',
    scope)
client = gspread.authorize(creds)

# Abre la hoja
sheet_id = 'TU_SHEET_ID_AQUI'
sheet = client.open_by_key(sheet_id).worksheet('Ventas') # Nombre de la hoja

# Extrae datos a DataFrame
df = get_as_dataframe(sheet)

print("Datos extraídos:")
print(df.head())
```

Script sugerido

Listing 2: ETL con SQLAlchemy

```
import pandas as pd
from sqlalchemy import create_engine, Integer, String, Float, DateTime, func
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped_column, sessionmaker
from typing import Optional

# Simulación de Extracción de Google Sheets (reemplaza con gspread en producció
    n)
def extraer_datos():
    data = {
        'ID': [1, 2, 3, 4, 5],
        'Producto': ['Libro', 'Laptop', 'Teléfono', 'Silla', 'Mesa'],
        'Cantidad': [2, 1, 3, 4, 1],
        'Precio': [15.5, 800.0, 300.0, 50.0, 120.0],
        'Fecha': ['2025-01-10', '2025-02-15', '2025-03-20', '2025-04-25', '
           2025-05-30'],
        'Cliente_ID': [101, 102, 101, 103, 102]
```

```
    }
    df = pd.DataFrame(data)
    df['Fecha'] = pd.to_datetime(df['Fecha']) # Convierte a datetime
    return df

#Transformación
def transformar_datos(df):
    df = df.dropna() # Limpieza básica
    df['Total'] = df['Cantidad'] * df['Precio'] # Cálculo nuevo
    return df

# Definición de Modelos con SQLAlchemy (Énfasis en ORM Declarativo)
class Base(DeclarativeBase):
    pass

class Venta(Base):
    __tablename__ = 'ventas'
    id: Mapped[int] = mapped_column(Integer, primary_key=True)
    producto: Mapped[str] = mapped_column(String(100))
    cantidad: Mapped[int] = mapped_column(Integer)
    precio: Mapped[float] = mapped_column(Float)
    fecha: Mapped[DateTime] = mapped_column(DateTime)
    cliente_id: Mapped[int] = mapped_column(Integer)
    total: Mapped[Optional[float]] = mapped_column(Float, nullable=True)

# Función Principal de ETL
def main():
    # Extracción
    df = extraer_datos()

    # Transformación
    df = transformar_datos(df)

    # Conexión y Creación de BD con SQLAlchemy
    engine = create_engine('sqlite:///ventas.db', echo=True) # echo para
        logging
    Base.metadata.create_all(engine) # Crea tablas si no existen

    # Sesión Factory
    Session = sessionmaker(bind=engine)

    # Carga (Load)

    ventas = []
    for _, row in df.iterrows():
        venta = Venta(
            id=int(row['ID']),
            producto=row['Producto'],
            cantidad=int(row['Cantidad']),
            precio=row['Precio'],
            fecha=row['Fecha'],
```

```
            cliente_id=int(row['Cliente_ID']),
            total=row['Total']
        )
    ventas.append(venta)

    with Session() as session:
        try:
            session.add_all(ventas)
            session.commit()
            print("Datos cargados exitosamente en ventas.db.")
        except Exception as e:
            session.rollback()
            print(f"Error en carga: {e}")
    # Ejecución de Consultas con ORM
    with Session() as session:
        # Consulta 1: Todas las ventas
        todas_ventas = session.query(Venta).all()
        print("\nTodas las ventas.")
        for v in todas_ventas:
            print(f"ID: {v.id}, Producto: {v.producto}, Total: {v.total}")

        # Consulta 2: Total de ventas después de una fecha (filtro y agregación
            )
        total_ventas = session.query(func.sum(Venta.total)).filter(Venta.fecha
            > pd.to_datetime("2025-03-01")).scalar()
        print(f"\nTotal de ventas después de 2025-03-01: {total_ventas}")

        # Consulta 3: Ventas por cliente (group by implicito en query)
        ventas_por_cliente = session.query(Venta.cliente_id, func.count(Venta.
            id)).group_by(Venta.cliente_id).all()

        print("\nVentas por cliente:")
        for cliente, count in ventas_por_cliente:
            print(f"Cliente ID: {cliente}, Número de ventas: {count}")


    if __name__ == "__main__":
        main()
```