Carga de Datos a Destinos - Día 4

Change Status

in-progress 40 min

Learning Objectives

1. Implementar estrategias eficientes de carga a bases de datos SQL usando to_sql()
2. Comprender diferencias entre carga completa vs incremental
3. Gestionar integridad referencial y conflictos durante la carga

Theory | Practice | Quiz | Evidence

Ejercicio: Implementar carga completa con validaciones y estrategias avanzadas

Ejercicio práctico para aplicar los conceptos aprendidos.

Crear esquema de base de datos destino:

```
import sqlite3
import pandas as pd
import numpy as np

# Crear base de datos
conn = sqlite3.connect('ventas_etl.db')

# Crear tablas con constraints
conn.execute('''
    CREATE TABLE clientes (
        id_cliente INTEGER PRIMARY KEY,
        nombre TEXT NOT NULL,
        email TEXT UNIQUE,
        ciudad TEXT,
        fecha_registro DATE
    )
''')
```

```
conn.execute('''
    CREATE TABLE productos (
        id_producto INTEGER PRIMARY KEY,
        nombre TEXT NOT NULL,
        precio REAL NOT NULL,
        categoria TEXT
    )
''')

conn.execute('''
    CREATE TABLE ventas (
        id_venta INTEGER PRIMARY KEY,
        id_cliente INTEGER,
        id_producto INTEGER,
        cantidad INTEGER NOT NULL,
        precio_unitario REAL NOT NULL,
        fecha_venta DATE,
        FOREIGN KEY (id_cliente) REFERENCES clientes(id_cliente),
        FOREIGN KEY (id_producto) REFERENCES productos(id_producto)
    )
''')

conn.commit()
```

Crear datos de ejemplo para carga:

```
# Datos de clientes
clientes_df = pd.DataFrame({
    'id_cliente': range(1, 6),
    'nombre': ['Ana García', 'Carlos López', 'María Rodríguez', 'Juan Pérez', 'Luis Martín'],
    'email': ['ana@email.com', 'carlos@email.com', 'maria@email.com', 'juan@email.com', 'luis@email.com'],
    'ciudad': ['Madrid', 'Barcelona', 'Madrid', 'Valencia', 'Sevilla'],
    'fecha_registro': pd.date_range('2023-01-01', periods=5, freq='MS')
})

# Datos de productos
productos_df = pd.DataFrame({
    'id_producto': range(101, 106),
    'nombre': ['Laptop', 'Mouse', 'Teclado', 'Monitor', 'Audífonos'],
    'precio': [1200, 25, 80, 300, 150],
    'categoria': ['Electrónica', 'Accesorios', 'Accesorios', 'Electrónica', 'Audio']
})

# Datos de ventas (con algunos errores intencionales)
```

```
np.random.seed(42)
ventas_df = pd.DataFrame({
    'id_venta': range(1, 21),
    'id_cliente': np.random.choice(range(1, 8), 20),  # Algunos IDs inexistentes
    'id_producto': np.random.choice(range(101, 108), 20),  # Algunos IDs inexistentes
    'cantidad': np.random.randint(1, 5, 20),
    'precio_unitario': np.random.choice([1200, 25, 80, 300, 150], 20),
    'fecha_venta': pd.date_range('2024-01-01', periods=20, freq='D')
})
```

Implementar carga con validaciones:

```
# Función para cargar con validaciones
def cargar_con_validacion(df, tabla, conn, claves_foraneas=None):
    try:
        # Validar claves foráneas si se especifican
        if claves_foraneas:
            for columna, tabla_ref, columna_ref in claves_foraneas:
                valores_validos = pd.read_sql(f'SELECT {columna_ref} FROM {tabla_ref}', conn)
                valores_validos = valores_validos[columna_ref].tolist()

                invalidos = ~df[columna].isin(valores_validos)
                if invalidos.any():
                    print(f"Advertencia: {invalidos.sum()} registros en {columna} no existen en {tabla_ref}")
                    # Opción: filtrar inválidos o marcar como NULL
                    df = df[~invalidos]  # Filtrar inválidos

        # Cargar datos
        df.to_sql(tabla, conn, index=False, if_exists='append')
        print(f"✓ Cargados {len(df)} registros en {tabla}")
        return True

    except Exception as e:
        print(f"✗ Error cargando {tabla}: {e}")
        return False

# Cargar tablas base (sin dependencias)
exito_clientes = cargar_con_validacion(clientes_df, 'clientes', conn)
exito_productos = cargar_con_validacion(productos_df, 'productos', conn)

# Cargar ventas con validaciones de FK
if exito_clientes and exito_productos:
    claves_ventas = [
        ('id_cliente', 'clientes', 'id_cliente'),
        ('id_producto', 'productos', 'id_producto')
```

```
    ]
    cargar_con_validacion(ventas_df, 'ventas', conn, claves_ventas)
```

Verificar carga y ejecutar consultas:

```
# Verificar conteos
for tabla in ['clientes', 'productos', 'ventas']:
    count = pd.read_sql(f'SELECT COUNT(*) FROM {tabla}', conn).iloc[0,0]
    print(f"{tabla}: {count} registros")

# Consulta de ejemplo: ventas por cliente
query_result = pd.read_sql('''
    SELECT c.nombre, COUNT(v.id_venta) as num_ventas,
           SUM(v.cantidad * v.precio_unitario) as total_ventas
    FROM clientes c
    LEFT JOIN ventas v ON c.id_cliente = v.id_cliente
    GROUP BY c.id_cliente, c.nombre
    ORDER BY total_ventas DESC
''', conn)

print("\nVentas por cliente:")
print(query_result)

conn.close()
```

Verificación: Confirma que todas las validaciones funcionaron correctamente y que solo se cargaron datos válidos con integridad referencial intacta.

Requerimientos:

Python con Pandas y sqlite3
Conocimiento de SQL básico
Base de datos SQLite para pruebas