# Carga de Datos y Estrategias de Destino - Día 4

Change Status

in-progress 40 min

## Learning Objectives

1. Entender estrategias de carga de datos
2. Aprender carga a diferentes tipos de bases de datos
3. Comprender diferencias entre carga completa e incremental
4. Conocer optimizaciones de performance en carga



## Activities and Learning

### Task 1: Estrategias de Carga (10 minutos)

**¿Cómo cargar datos eficientemente?**

Existen diferentes estrategias dependiendo del volumen de datos y frecuencia de actualización.

**Tipos de carga:**

**Completa (Full Load)**: Reemplazar todos los datos
**Incremental**: Solo actualizar cambios
**Upsert**: Insertar nuevos, actualizar existentes
**Streaming**: Carga continua en tiempo real

**Cuándo usar cada estrategia:**

**Full Load**: Datasets pequeños, cambios masivos
**Incremental**: Grandes volúmenes, cambios frecuentes
**Upsert**: Datos con claves naturales
**Streaming**: Requisitos de baja latencia

### Task 2: Carga a Bases de Datos SQL (10 minutos)

**¿Cómo escribir datos a bases de datos?**

La carga a SQL requiere considerar constraints, índices y performance.

**Consideraciones importantes:**

**Transacciones**: Agrupar operaciones para consistencia
**Batch size**: Tamaño óptimo de lotes
**Error handling**: Qué hacer con registros problemáticos
**Índices**: Desactivar durante carga masiva

**Ejemplo de carga SQL:**

```
import pandas as pd
from sqlalchemy import create_engine

def cargar_a_postgresql(df, tabla, engine):
    try:
        # Cargar datos
        df.to_sql(
            tabla,
            engine,
            if_exists='append',  # 'replace' para full load
            index=False,
            chunksize=1000  # Batch size
        )
        print(f"Cargados {len(df)} registros a {tabla}")
        return True
    except Exception as e:
        print(f"Error en carga: {e}")
        return False
```

### Task 3: Carga a Formatos Analíticos (10 minutos)

**¿Cómo optimizar para análisis?**

Los formatos analíticos priorizan velocidad de lectura sobre escritura.

**Formatos eficientes:**

**Parquet**: Columnar, comprimido, rápido para analytics
**Delta Lake**: Transaccional, con historial de cambios
**Iceberg**: Formato abierto con evolución de esquemas

**Ejemplo de carga a Parquet:**

```
def cargar_a_parquet(df, ruta_archivo):
    try:
        df.to_parquet(
            ruta_archivo,
            engine='pyarrow',
            compression='snappy',  # Buena compresión + velocidad
            index=False
        )
        print(f"Datos guardados en {ruta_archivo}")
        return True
    except Exception as e:
        print(f"Error guardando Parquet: {e}")
        return False
```