Manejo de Errores y Logging en ETL - Día 5

Change Status

pending 40 min

Learning Objectives

1. Entender importancia del manejo de errores robusto
2. Aprender técnicas de logging efectivo
3. Comprender validación post-ETL
4. Conocer mejores prácticas para pipelines observables

Practical exercise to apply the concepts learned.

Ejercicio: Construir pipeline ETL con manejo de errores completo

Configurar logging:

```
import logging
import time
from functools import wraps

# Configurar logging
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(levelname)s - %(message)s',
    handlers=[
        logging.FileHandler('etl_ecommerce.log'),
        logging.StreamHandler()
    ]
)

logger = logging.getLogger('etl_ecommerce')

def log_etapa(etapa):
    """Decorator para logging de etapas"""
    def decorator(func):
        @wraps(func)
        def wrapper(*args, **kwargs):
            logger.info(f"🚀 Iniciando {etapa}")
            start_time = time.time()

            try:
                result = func(*args, **kwargs)
                duration = time.time() - start_time
                logger.info(f"✅ {etapa} completada en {duration:.2f}s")
                return result
            except Exception as e:
                duration = time.time() - start_time
                logger.error(f"💥 {etapa} falló en {duration:.2f}s: {e}")
                raise e

        return wrapper
    return decorator
```

Pipeline ETL con error handling:

```
import pandas as pd
import numpy as np
from typing import Dict, Any

class ETLPipeline:
    def __init__(self):
        self.logger = logger
        self.errores = []

    @log_etapa("extracción de datos")
    def extract(self) -> pd.DataFrame:
        """Extraer datos con manejo de errores"""
        try:
            # Simular extracción (podría fallar)
            if np.random.random() < 0.1:  # 10% chance de error
                raise ConnectionError("Error de conexión a fuente de datos")

            # Datos de ejemplo
            datos = pd.DataFrame({
                'orden_id': range(1, 101),
```

```
                'cliente_id': np.random.randint(1, 21, 100),
                'producto': np.random.choice(['A', 'B', 'C', 'D'], 100),
                'cantidad': np.random.randint(1, 6, 100),
                'precio': np.round(np.random.uniform(10, 200, 100), 2)
            })

            self.logger.info(f"Extraídos {len(datos)} registros")
            return datos

        except Exception as e:
            self.errores.append(f"Extract: {e}")
            raise e

    @log_etapa("transformación de datos")
    def transform(self, datos: pd.DataFrame) -> pd.DataFrame:
        """Transformar datos con validaciones"""
        try:
            df = datos.copy()

            # Validar datos de entrada
            if df.empty:
                raise ValueError("No hay datos para transformar")

            # Transformaciones
            df['total'] = df['cantidad'] * df['precio']
            df['categoria_precio'] = pd.cut(
                df['precio'],
                bins=[0, 50, 100, 200],
                labels=['Bajo', 'Medio', 'Alto']
            )

            # Validar transformaciones
            if df['total'].isnull().any():
                raise ValueError("Transformación produjo valores nulos")

            self.logger.info(f"Transformados {len(df)} registros")
            return df

        except Exception as e:
            self.errores.append(f"Transform: {e}")
            raise e

    @log_etapa("carga de datos")
    def load(self, datos: pd.DataFrame) -> bool:
        """Cargar datos con verificación"""
        try:
            # Simular carga (podría fallar)
            if np.random.random() < 0.05:  # 5% chance de error
                raise Exception("Error de conexión a base de datos")

            # En producción: datos.to_sql('ventas', engine, if_exists='append')
            self.logger.info(f"Cargados {len(datos)} registros exitosamente")

            # Validar carga
            registros_esperados = len(datos)
            registros_cargados = len(datos)  # Simulado

            if registros_cargados != registros_esperados:
                raise ValueError(f"Carga incompleta: {registros_cargados}/{registros_esperados}")

            return True

        except Exception as e:
            self.errores.append(f"Load: {e}")
            raise e

    def ejecutar_pipeline(self) -> Dict[str, Any]:
        """Ejecutar pipeline completo con manejo de errores"""
        self.logger.info("🎯 Iniciando pipeline ETL completo")

        try:
            # Extract
            datos_crudo = self.extract()

            # Transform
            datos_transformados = self.transform(datos_crudo)

            # Load
            exito = self.load(datos_transformados)

            resultado = {
                'exito': True,
                'registros_procesados': len(datos_transformados),
                'errores': self.errores
            }

            self.logger.info("🎉 Pipeline ETL completado exitosamente")
            return resultado

        except Exception as e:
            self.logger.error(f"💥 Pipeline ETL falló: {e}")

            return {
                'exito': False,
                'error_principal': str(e),
```

```
            'errores': self.errores
        }
```

Ejecutar y validar pipeline:

```
# Ejecutar pipeline con diferentes escenarios
pipeline = ETLPipeline()

# Ejecución exitosa
resultado = pipeline.ejecutar_pipeline()

print(f"\nResultado del pipeline:")
print(f"Éxito: {resultado['exito']}")
if resultado['exito']:
    print(f"Registros procesados: {resultado['registros_procesados']}")
else:
    print(f"Error principal: {resultado['error_principal']}")

print(f"Errores registrados: {len(resultado['errores'])}")
for error in resultado['errores']:
    print(f"  - {error}")

# Ejecutar múltiples veces para probar robustez
resultados_multiples = []
for i in range(5):
    print(f"\n--- Ejecución {i+1} ---")
    pipeline_i = ETLPipeline()
    resultado_i = pipeline_i.ejecutar_pipeline()
    resultados_multiples.append(resultado_i['exito'])

exito_rate = sum(resultados_multiples) / len(resultados_multiples)
print(".1%")
```

Verificación: ¿Qué información debería incluir en los logs para facilitar el debugging? ¿Cómo decides entre continuar el pipeline con errores parciales vs detenerlo completamente?

Requerimientos:

Logging module de Python
Pandas para manipulación de datos
Conocimiento de manejo de excepciones