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

Activities and Learning

Task 1: Estrategias de Manejo de Errores (10 minutos)

¿Por qué fallan los pipelines ETL?

Los pipelines pueden fallar por múltiples razones: datos corruptos, conexiones caídas, recursos insuficientes, lógica errónea.

Tipos de errores comunes:

Datos: Valores faltantes, formatos inválidos, duplicados
Conectividad: APIs caídas, bases de datos inaccesibles
Recursos: Memoria insuficiente, timeouts
Lógica: Errores en transformaciones, cálculos incorrectos

Estrategias de manejo:

```
def procesar_con_errores(operacion, max_reintentos=3):
    """Template para manejar errores con reintentos"""
    for intento in range(max_reintentos):
        try:
            return operacion()
        except Exception as e:
            if intento == max_reintentos - 1:
                print(f"Error definitivo: {e}")
                raise e
            else:
                print(f"Intento {intento + 1} falló: {e}. Reintentando...")
                time.sleep(2 ** intento)  # Exponential backoff
```

Task 2: Logging Efectivo (10 minutos)

¿Cómo hacer logging útil?

El logging debe ayudar a debuggear problemas y monitorear el estado del pipeline.

Niveles de logging:

DEBUG: Información detallada para desarrollo
INFO: Eventos normales del pipeline
WARNING: Situaciones que requieren atención
ERROR: Errores que impiden funcionamiento
CRITICAL: Errores que requieren intervención inmediata

Ejemplo de logging configurado:

```
import logging

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

logger = logging.getLogger('etl_pipeline')

# Uso en pipeline
def etapa_etl(datos):
```

```
    logger.info(f"Iniciando procesamiento de {len(datos)} registros")

    try:
        # Procesamiento
        resultado = procesar_datos(datos)
        logger.info(f"Procesamiento completado: {len(resultado)} registros")
        return resultado

    except Exception as e:
        logger.error(f"Error en procesamiento: {e}")
        raise e
```

Task 3: Validaciones Post-ETL (10 minutos)

¿Cómo verificar que el ETL funcionó correctamente?

Las validaciones post-ETL aseguran que los datos cargados sean correctos y completos.

Validaciones importantes:

Conteo de registros: ¿Llegaron todos los datos?
Suma de controles: ¿Los totales coinciden?
Calidad de datos: ¿Se mantuvieron las reglas?
Integridad referencial: ¿Las relaciones son válidas?

Ejemplo de validaciones:

```
def validar_carga(origen_df, destino_tabla, engine):
    """Validar que la carga fue exitosa"""
    validaciones = {}

    # Contar registros
    origen_count = len(origen_df)
    destino_count = pd.read_sql(f"SELECT COUNT(*) FROM {destino_tabla}", engine).iloc[0, 0]
    validaciones['conteo_registros'] = origen_count == destino_count

    # Suma de totales
    origen_total = origen_df['total'].sum()
    destino_total = pd.read_sql(f"SELECT SUM(total) FROM {destino_tabla}", engine).iloc[0, 0]
    validaciones['suma_totales'] = abs(origen_total - destino_total) < 0.01

    # Valores faltantes
    faltantes = pd.read_sql(f"SELECT COUNT(*) FROM {destino_tabla} WHERE total IS NULL", engine).iloc[0, 0]
    validaciones['sin_faltantes'] = faltantes == 0

    return validaciones
```