ANÁLISIS SNOWFLAKE ETL

Proyecto Integrador - Avance 3

Autor: Facundo Acosta

Fecha: 10/15/2025

INDICE

1. Resumen Ejecutivo y Contexto del Negocio

El objetivo de este tercer avance de nuestro PI es evolucionar de un sistema transaccional en **PostgreSQL**, enfocado en la operación diaria, a un **Data Warehouse analítico en Snowflake**. Esta transición es fundamental para que la dirección obtenga una visión estratégica a largo plazo, permitiendo el análisis de desempeño histórico, la identificación de tendencias y la evaluación de indicadores clave de negocio (KPIs).

Para lograrlo, se diseñó un **modelo en estrella** con una tabla de hechos central (FACT_DELIVERIES) y múltiples tablas de dimensiones (dim_date, dim_vehicle, dim_driver, etc.). La alimentación de este Data Warehouse se realiza mediante un **pipeline ETL automatizado en Python** que extrae, transforma y carga los datos.

Durante la implementación del pipeline, surgió un desafío crítico que amenazaba la integridad de toda la solución: la **duplicación de registros** al ejecutar el proceso de carga repetidamente. Este documento detalla la implementación de una **estrategia UPSERT** robusta que erradicó este problema, garantizando la fiabilidad y consistencia del Data Warehouse.

2. Arquitectura General del Sistema

El flujo de datos sigue una arquitectura ETL clásica, moviendo la información desde la base de datos operacional hacia el entorno analítico.

- **Extracción (Extract):** Un módulo de Python (FA_extract.py) se conecta a la base de datos PostgreSQL y extrae los datos operativos, generando archivos Parquet en un área de staging.
- **Transformación (Transform):** El módulo FA_transform.py toma los datos brutos, realiza validaciones de calidad, calcula métricas de negocio cruciales (como eficiencia de combustible y puntualidad) y prepara los datos para el modelo dimensional.
- **Carga (Load):** El módulo FA_load.py se conecta a Snowflake y es responsable de cargar los datos transformados. Es en esta fase donde se implementó la lógica UPSERT para prevenir duplicados.

3. El Desafío: La Amenaza de los Datos Duplicados

El problema principal identificado fue que el pipeline original carecía de un mecanismo de verificación. Al ejecutar el proceso de carga varias veces con la misma fuente de datos, se insertaban registros idénticos en la tabla FACT_DELIVERIES.

Esto comprometía gravemente la integridad de los datos, ya que cualquier análisis o reporte basado en esta tabla arrojaría resultados incorrectos (ej. ingresos inflados, distancias duplicadas). El objetivo se centró en transformar el pipeline en un proceso **idempotente**: sin importar cuántas veces se ejecute con los mismos datos, el estado final de la base de datos siempre debe ser el mismo y correcto.

4. La Solución: Implementación de una Estrategia UPSERT Dual

Para resolver el problema de la duplicación, se implementó una estrategia **UPSERT** (unión de UPDATE e INSERT) que gestiona de forma inteligente los registros. El sistema primero identifica si un registro ya existe y luego decide si debe actualizarlo o insertar uno nuevo.

La solución se desarrolló en tres pasos clave:

Paso 4.1: Identificación y Clasificación de Registros

Antes de escribir en Snowflake, el pipeline realiza un análisis previo:

1. **Consulta a Snowflake:** Se obtiene una lista de todos los DELIVERY_ID que ya existen en la tabla FACT_DELIVERIES.
2. **Comparación:** Esta lista se cruza con los DELIVERY_ID del nuevo lote de datos a procesar.
3. **Clasificación:** Los datos de entrada se dividen en dos DataFrames distintos:
    - **Registros Nuevos:** Aquellos cuyo DELIVERY_ID no existe en la tabla de destino (para INSERT).
    - **Registros a Actualizar:** Aquellos cuyo DELIVERY_ID ya existe (para UPDATE).

Paso 4.2: Estrategia de Carga Dual para Máxima Robustez

Para garantizar que el proceso de carga siempre se complete con éxito, se implementaron dos métodos complementarios.

- Método Principal: Comando MERGE Nativo de Snowflake

Este es el enfoque más eficiente y preferido. El comando MERGE realiza la comparación y la operación de inserción/actualización en una única transacción atómica, optimizando el rendimiento.

```
SQL:
MERGE INTO FACT_DELIVERIES AS target
USING tabla_temporal AS source
ON target.DELIVERY_ID = source.DELIVERY_ID
WHEN MATCHED THEN UPDATE SET todos_los_campos
WHEN NOT MATCHED THEN INSERT todos_los_campos
```

- Método de Respaldo (Fallback): DELETE + INSERT

Si el comando MERGE falla por cualquier motivo (ej. un cambio inesperado en la configuración de Snowflake), el sistema activa automáticamente un plan B universal y robusto.

1. **DELETE:** Se eliminan de FACT_DELIVERIES todos los registros cuyos DELIVERY_ID coinciden con los del lote "a actualizar"[25]. Para evitar límites técnicos, esta operación se realiza por lotes de 500 registros.
2. **INSERT:** A continuación, se insertan todos los registros del lote de entrada (tanto los nuevos como los que acaban de ser eliminados para su actualización).

Esta estrategia dual asegura una **máxima confiabilidad** en el proceso de carga.

Paso 4.3: Mecanismos de Robustez Adicionales

Para fortalecer aún más el pipeline, se implementaron varias mejoras:

- **Manejo de Tipos de Datos:** Se creó una función para convertir de forma segura los tipos de datos de Pandas (ej. numpy.int64) a tipos nativos de Python compatibles con SQL, evitando errores comunes.
- **Procesamiento por Lotes:** Las cargas y eliminaciones masivas se gestionan en bloques (batches) para optimizar el uso de memoria y los tiempos de transacción.
- **Logging Detallado:** Se configuró un sistema de logging que registra cada paso del proceso en un archivo etl_pipeline.log, facilitando el monitoreo y la depuración.

5. Resultados y Métricas Obtenidas

La implementación de la estrategia UPSERT fue exitosa y arrojó resultados medibles y positivos.

- **Eficiencia del Pipeline:** El tiempo promedio de ejecución del ETL completo, desde la extracción hasta la carga final, fue de **48-49 segundos**.
- **Integridad de Datos Garantizada:** Se procesaron y cargaron **1,898 registros** sin generar un solo duplicado. Una consulta de verificación en Snowflake confirmó el resultado:

```
SQL:

-- Resultado: total_registros = delivery_ids_unicos = 1,898

SELECT COUNT(*) as total_registros, COUNT(DISTINCT "DELIVERY_ID") as
delivery_ids_unicos

FROM "FACT_DELIVERIES";
```

- **Impacto de Negocio:** Con la seguridad de tener datos limpios y confiables, se calcularon métricas de negocio precisas:
    - **Entregas a tiempo:** 32.9%
    - **Eficiencia de combustible promedio:** 2.7 km/L
    - **Rentabilidad total analizada:** $289,075.88

6. Herramienta de Verificación y Consulta: FA_snowflake_verify_3

Para complementar el pipeline ETL y garantizar la calidad de los datos cargados en Snowflake, se desarrolló una **herramienta interactiva de consulta y verificación** (FA_snowflake_verify_3.py) que permite a los usuarios explorar y validar los datos del Data Warehouse de manera segura y eficiente.

Características Principales de la Herramienta:

- **Detección Automática de Columnas:** La herramienta detecta automáticamente las columnas disponibles en la tabla FACT_DELIVERIES, adaptándose dinámicamente a cambios en el esquema.
- **Menú Interactivo con Múltiples Opciones:** Ofrece 10 opciones predefinidas para consultas comunes, incluyendo:
    - Estructura de la tabla
    - Consultas rápidas de registros
    - Análisis de entregas por fecha
    - Eficiencia de combustible por vehículo
    - Rendimiento de conductores
    - Métricas de negocio principales
    - Verificación de integridad de datos
    - Análisis de rutas
    - Consultas personalizadas
    - Modo SQL interactivo
- **Mecanismos de Seguridad:**
    - Límites automáticos en consultas para prevenir sobrecargas
    - Manejo robusto de errores y conexiones
    - Validación de sintaxis SQL

- **Verificación de Autoria:** Incluye protección de derechos de autor y verificación de autoría del código.

Integración con el Pipeline ETL:

Esta herramienta funciona como **capa final de verificación** del proceso ETL, permitiendo:

1. Validar que los datos se hayan cargado correctamente
2. Verificar la ausencia de duplicados
3. Calcular métricas de negocio en tiempo real
4. Realizar análisis ad-hoc sin necesidad de conocimientos técnicos avanzados

La herramienta se conecta directamente al esquema STAR_SCHEMA en Snowflake utilizando las mismas credenciales configuradas para el proceso ETL, garantizando coherencia en el acceso a los datos.

7. Lecciones Aprendidas y Conclusiones

Este avance no solo resolvió un problema técnico crítico, sino que también proporcionó valiosos aprendizajes.

Aprendizajes Técnicos Clave:

1. **El Poder de MERGE:** El comando MERGE nativo de Snowflake es la herramienta más eficiente y limpia para operaciones UPSERT.
2. **La Importancia del Fallback:** En la ingeniería de datos, tener una estrategia de respaldo robusta es tan importante como el método principal.
3. **Manejo Explícito de Tipos de Datos:** La conversión de tipos entre diferentes sistemas (Pandas, Python, SQL) es una fuente común de errores sutiles y debe manejarse explícitamente.

Conclusión del Proyecto:

El pipeline ETL de FleetLogix se ha transformado en una solución **completa, robusta y escalable**. El sistema ahora garantiza la integridad de los datos en el Data Warehouse al diferenciar

automáticamente entre registros nuevos y existentes, eliminando por completo el riesgo de duplicados.

Con esta base sólida, FleetLogix está preparada para soportar análisis avanzados, desarrollar dashboards ejecutivos y fomentar una cultura de toma de decisiones basada en datos confiables.

Apéndice A: Estructura de Archivos del Proyecto

```
05_etl_pipeline/
│
├── src/
│   ├── FA_main.py                # Orquestador principal ETL
│   ├── FA_extract.py             # Extracción PostgreSQL
│   ├── FA_transform.py           # Transformación y calidad
│   ├── FA_load.py                # Carga Snowflake con UPSERT
│   └── FA_snowflake_verify_3.py  # Consultas y verificación Snowflake
│
├── config/
│   └── settings.ini              # Configuraciones de conexión
│
├── data/
│   └── staging/                  # Datos intermedios Parquet
│
└── logs/
    └── etl_pipeline.log          # Logging de ejecuciones
```