Federico Ceballos Torres
Avance 1 – Módulo 2
Data Science – SoyHenry

Proyecto Integrador - Fleetlogix

Índice

1. Introducción

Fleetlogix es una empresa de transporte y logística que opera una flota de 200 vehículos realizando entregas de última milla en cinco ciudades principales de Colombia. El objetivo del proyecto es migrar de sistemas legacy hacia una infraestructura moderna de datos, basada en PostgreSQL, que permita análisis operativos y decisiones en tiempo real.

El proyecto integra modelado relacional, generación de datos sintéticos masivos, validaciones de calidad, KPIs logísticos y una arquitectura cloud escalable. La solución se diseñó siguiendo principios de reproducibilidad, integridad referencial y escalabilidad.

2. Objetivos técnicos

- Poblar una base de datos PostgreSQL con más de 500.000 registros sintéticos generados mediante librerías de Python.
- Garantizar integridad referencial mediante claves primarias y foráneas, restricciones de unicidad y validaciones de negocio.
- Documentar el modelo relacional con un diagrama ERD profesional y un modelo dimensional tipo estrella para OLAP.
- Implementar queries SQL para validaciones de calidad y KPIs operativos.
- Proponer una arquitectura cloud con Kafka, Flink/Spark, Data Lake y Power BI para análisis en tiempo real.

3. Modelo Relacional (ERD)

El modelo relacional se compone de seis tablas:

- **3.1. Maestras:**
  - vehicles: atributos técnicos (placa, tipo, capacidad, estado).
  - drivers: información personal y licencias.
  - routes: rutas entre ciudades con distancia y duración estimada.

- **3.2. Transaccionales:**
  - trips: viajes realizados, vinculando vehículo, conductor y ruta.
  - deliveries: entregas asociadas a cada viaje, con tracking único.
  - maintenance: registros de mantenimiento por vehículo.

- **3.3 Restricciones técnicas:**
  - Claves primarias (PK) en todas las tablas.
  - Claves foráneas (FK) para garantizar integridad referencial.
  - Restricciones de unicidad en placas, licencias y tracking numbers.
  - Índices en campos críticos (vehicle_id, driver_id, route_id) para optimizar queries.

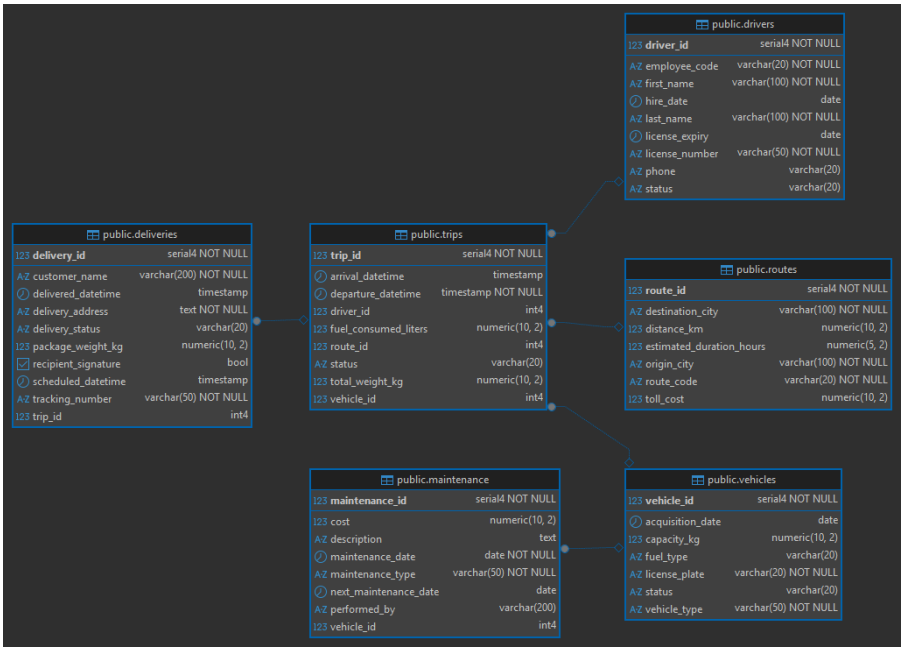

Figura 1. Diagrama Entidad–Relación (ERD) de Fleetlogix.
Se muestran las tablas maestras y transaccionales con sus relaciones y restricciones.

4. Generación de Datos Sintéticos

4.1. Herramientas: Python + Faker + pandas + numpy + psycopg2.

4.2. Volumen generado:

- 200 vehículos
- 400 conductores
- 50 rutas
- 100.000 viajes
- 400.001 entregas*
- 5.000 mantenimientos

4.3. Métodos clave:

- generate_trips(): simula viajes con distribución horaria, consumo de combustible y carga coherente.
- get_hourly_distribution(): concentra actividad entre 6:00–20:00, con picos en 8–12 y 14–18.
- Generación probabilística controlada: asegura realismo en entregas por viaje (2–6) y peso ≤ capacidad.

4.4. ¿Por qué 400.001 y no 400.000?

La razón principal es la generación probabilística. El generador de datos no asigna un número fijo de entregas por viaje, sino que usa una distribución probabilística controlada. Cada viaje (trip) recibe entre 2 y 6 entregas según una probabilidad definida en el método generate_trips(). Supongamos que el generador asigna entregas con una media de 4 por viaje. Para 100 000 viajes, el valor esperado es 400 000 entregas. Pero como se usa aleatoriedad (probabilidades), algunos viajes pueden tener 5 o 6 entregas y otros 2 o 3. Esa variación hace que el total final sea de 400 001, un desvío mínimo respecto al valor esperado.

4.5. Aspecto técnico

El generador usa funciones como random.choice() o distribuciones de numpy/Faker.
Estas funciones producen resultados enteros en cada iteración, y al acumularlos se obtiene un total que puede diferir en +-1, +-2, etc. El hecho de que haya salido 400 001 refleja la naturaleza aleatoria controlada del proceso.

En estadística, esto se llama fluctuación alrededor del valor esperado.

5. Validaciones de Calidad

5.1 Integridad referencial

Que todas las claves foráneas (FK) sean válidas.

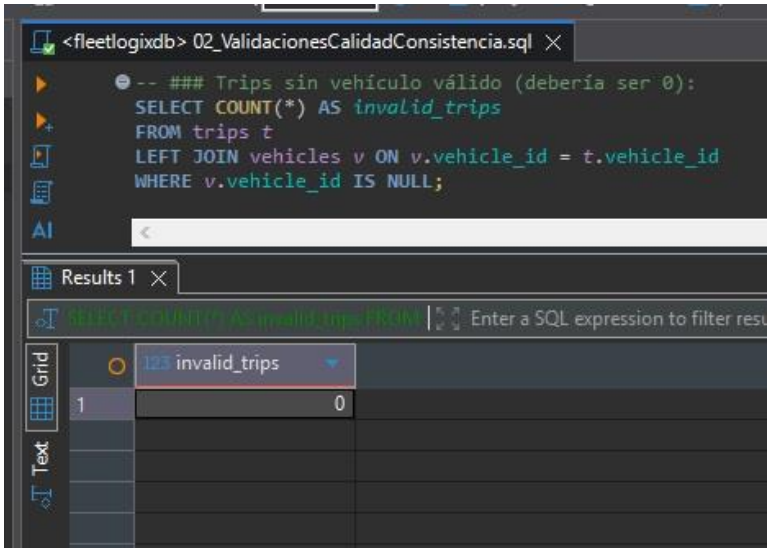

Figura 2. Ej.: Validación de integridad referencial en trips.
No existen viajes sin vehículo asociado, resultado esperado: 0.

5.2 Consistencia temporal

arrival_datetime > departure_datetime.

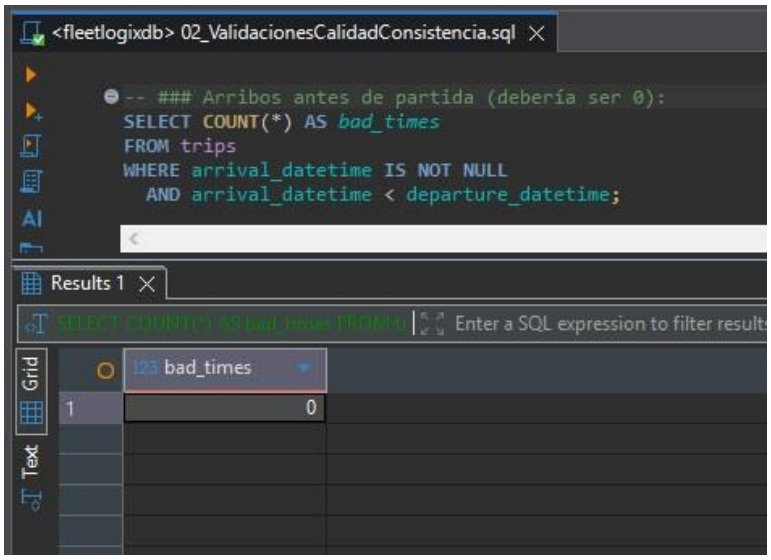

Figura 3. Ej.: Validación de consistencia temporal en trips.
No existen viajes con hora de llegada anterior a la de partida.

5.3 Reglas de negocio

- Peso ≤ capacidad del vehículo
- Tracking numbers únicos
- Entregas por viaje entre 2 y 6

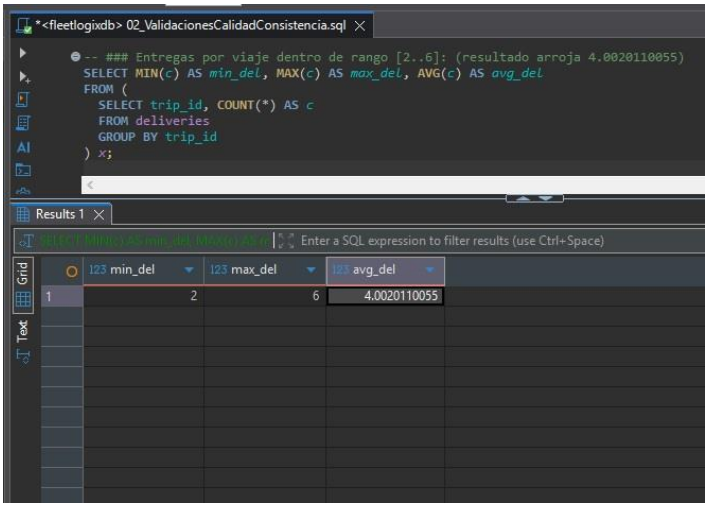

Figura 4. Ej.: Validación de entregas por viaje.
El promedio de entregas por viaje es 4, dentro del rango esperado [2–6].

6. KPIs Operativos

6.1 Métricas calculadas

- Porcentaje de entregas a tiempo vs retrasadas.
- Consumo promedio de combustible por tipo de vehículo.
- Utilización de capacidad por viaje.
- Mantenimientos por cada 1.000 km.
- Promedio de entregas por viaje.

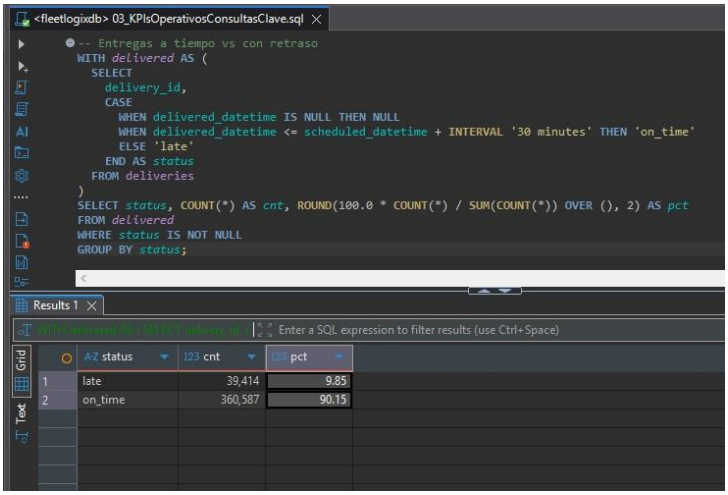

Figura 5. Ej.: KPI de desempeño en entregas.
El 90.15% de las entregas se realizaron a tiempo, el 9.85% con retraso.

7. Arquitectura Cloud Propuesta

- **Ingesta:** PostgreSQL + CDC (Debezium)
- **Streaming:** Apache Kafka
- **Procesamiento:** Apache Flink / Spark Streaming
- **Almacenamiento:** Data Lake + Data Warehouse
- **Visualización:** Power BI / Looker
- **Gobernanza:** Data Catalog + Lineage

La arquitectura cloud de Fleetlogix funciona como un flujo continuo que integra captura, transmisión, procesamiento, almacenamiento y visualización de datos. Los cambios en PostgreSQL se detectan en tiempo real mediante CDC y se envían a través de Kafka, que asegura distribución confiable y escalable. Flink o Spark Streaming procesan los eventos de manera paralela, aplicando reglas y cálculos sobre grandes volúmenes. Los resultados se guardan en un Data Lake para históricos y en un Data Warehouse para análisis. Finalmente, dashboards interactivos permiten decisiones rápidas, con gobernanza que garantiza calidad y trazabilidad.

8. Ejecución

1ra parte:

a. Abrir PowerShell.

b. Ir a la ubicación local de los scripts ejemplo: cd "RUTA\LOCAL\DE\LA\CARPETA"

c. .\run_fleetlogix.ps1

Si solicita contraseña es la definida en el script .py (puede pedirla una vez antes de crear la DB y otra vez antes de crear el esquema, es la misma ambas veces ya que es la del usuario).
Aquí abajo, tres capturas de pantalla del proceso finalizado:

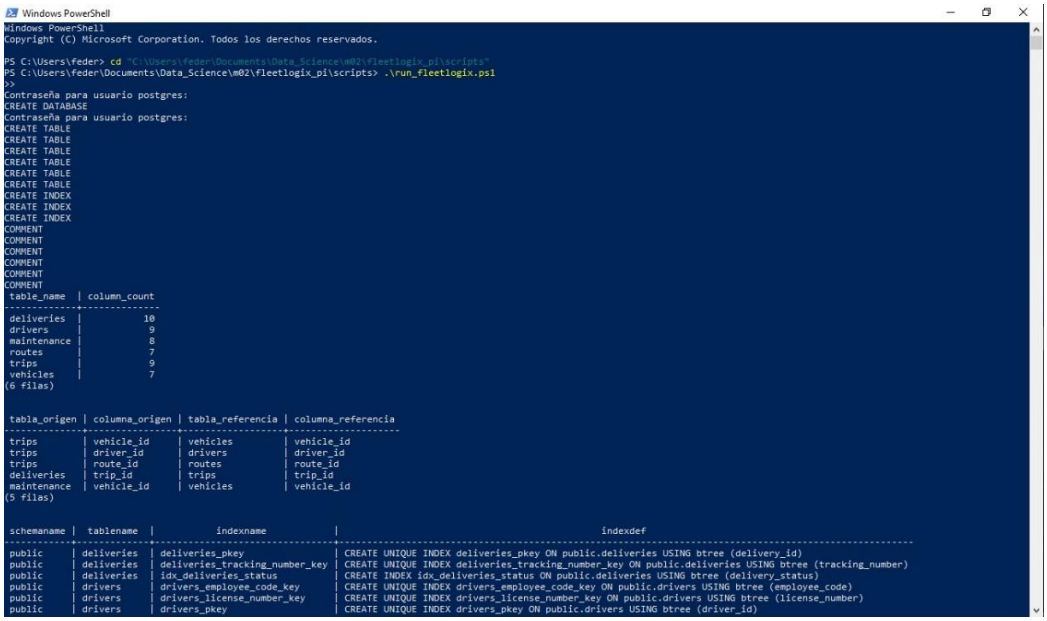

Figura 6. Ejecución del script run_fleetlogix.ps1
La consola muestra la creación de la base de datos, tablas, índices y comentarios.

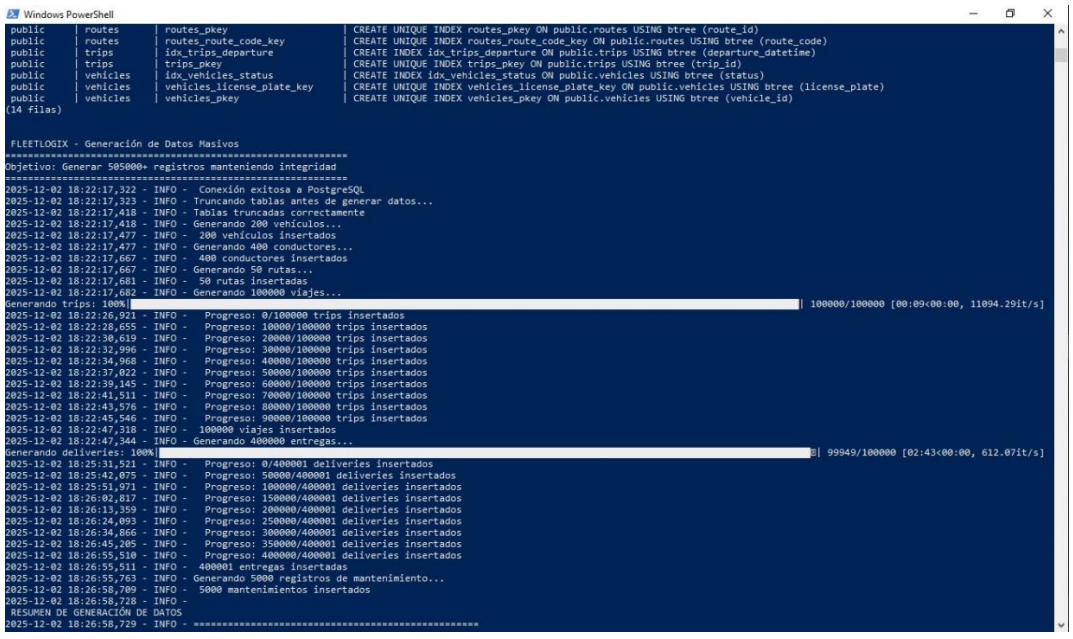

Figura 7. Generación masiva de datos sintéticos.
Se observa la inserción de más de 505.000 registros en PostgreSQL con logs de progreso.

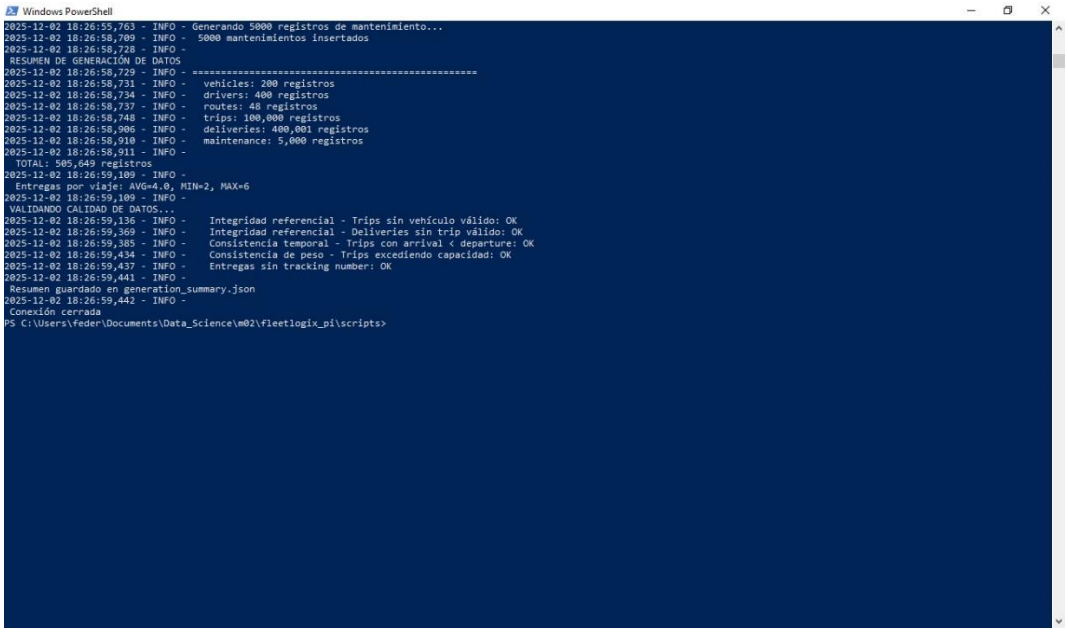

Figura 8. Resumen de generación y validaciones automáticas.
El log confirma integridad referencial, consistencia temporal y unicidad de claves.

2da parte:

4. Crear una nueva conexión para visualizar el esquema en DBeaver:
- Database -> New Database Connection -> PostgreSQL ->
- Rellenar con los parámetros definidos en el script:
Host:localhost Port:5432 Database:fleetlogixdb Username:postgres Password:fede0309

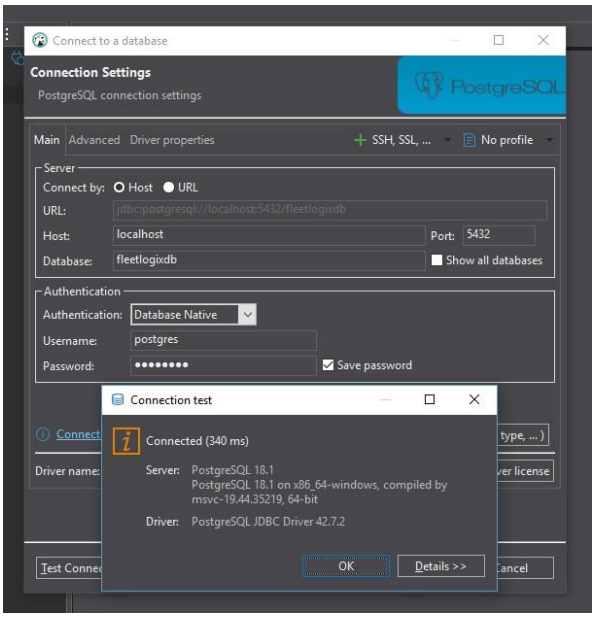

Figura 9. Conexión a la base fleetlogixdb en Dbeaver. Se confirma conexión exitosa con PostgreSQL 18.1 y la base está lista para la tercera parte de la ejecución.

3ra parte

5. Verificar carga de datos

- En la misma DB abrir y ejecutrar las siguientes queries en cualquier orden, una a una o todas juntas:

'01_InventarioDeTablasPKsFKsIndices'
'02_ValidacionesCalidadConsistencia.sql'
'03_KPIsOperativosConsultasClave.sql'

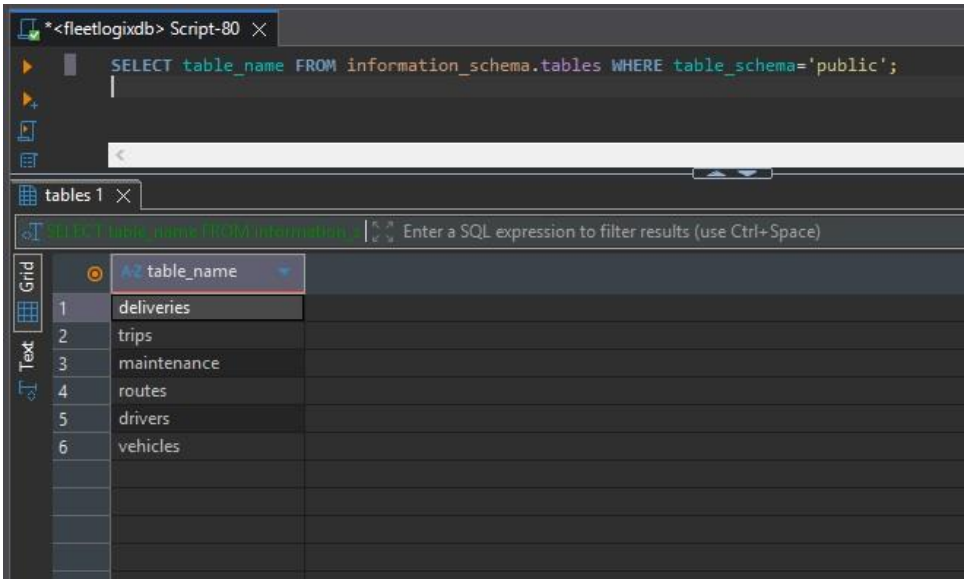

Figura 10. Inventario de tablas creadas en el esquema public.
Se listan las tablas: deliveries, trips, maintenance, routes, drivers y vehicles.

9. Conclusiones

El diagnóstico y ensayo sobre el proyecto Fleetlogix demuestra que es posible realizar una migración desde sistemas legacy hacia una infraestructura moderna de datos, garantizando calidad, integridad y consistencia. El modelo relacional soporta operaciones logísticas a gran escala, mientras que el modelo dimensional habilita análisis OLAP y dashboards. La arquitectura cloud propuesta asegura escalabilidad y decisiones en tiempo real.