Avance #1

Generación de Datos Sintéticos con Python y PostgreSQL

En el primer avance se generan más de **500.000 registros simulados** para poblar una base de datos PostgreSQL y simular la base de datos de la empresa de transporte Fleetlogix.

Fleetlogix es una empresa de transporte y logística que opera una flota de 200 vehículos realizando entregas de última milla en 5 ciudades principales. La empresa ha estado operando con sistemas legacy y hojas de cálculo, pero necesita modernizar su infraestructura de datos para competir en el mercado actual.

De sus sistemas podemos sacar las siguientes tablas:

- Vehículos
- Conductores
- Rutas
- Viajes
- Entregas
- Mantenimientos

El objetivo del primer avance es crear un dataset, coherente e integrado que permita realizar pruebas, análisis o construir dashboards.

Herramientas Utilizadas

- **Python 3.9+**
- **PostgreSQL 13+**
- Librerías de Python:
    - psycopg2
    - Faker
    - NumPy
    - pandas
    - tqdm
    - logging
    - json
    - DateTime
    - os
    - dotenv

Instalación del Motor PostgreSQL

1. Descargar PostgreSQL

Ir a la página oficial:

- https://www.enterprisedb.com/downloads/postgres-postgresql-downloads

Seleccionar tu sistema operativo e instalar:

- PostgreSQL Versión 17.7

Durante la instalación:

- Define una contraseña para el usuario “postgres”
- Deja por defecto el puerto “**5432”**

2. Crear la base de datos

- Abre Dbeaver y genera una nueva conexión con Ctrl + Shift + N
- Configura la conexión con PostgreSQL
- Deja la configuración de host igual
- Coloca tu contraseña y listo
- Crea un nuevo script SQL y ejecuta:

```
CREATE DATABASE fleetlogix_database;
```

3. Creación de esquemas y tablas

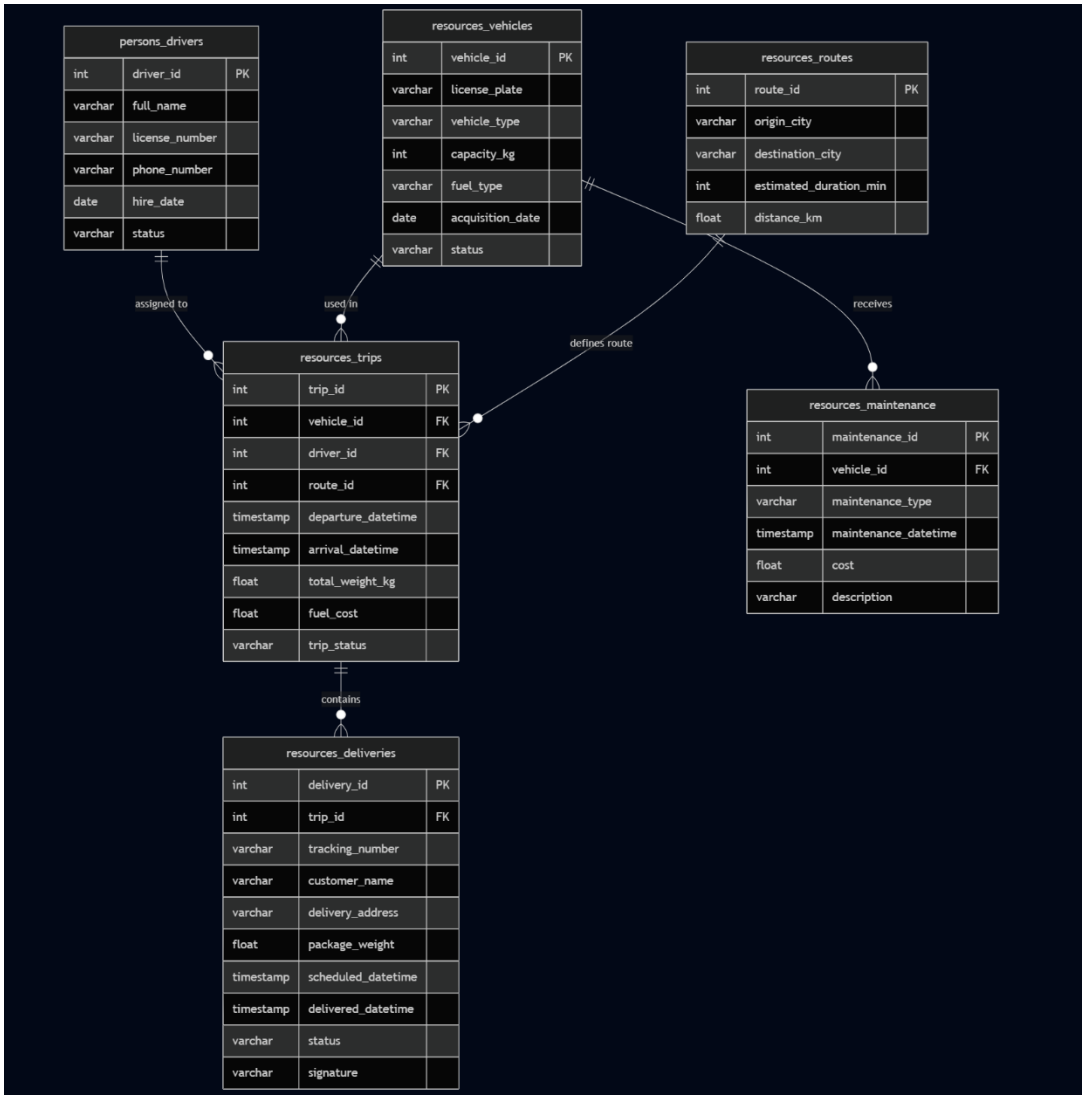
```
-- Esquema Person (Personas)

CREATE SCHEMA Persons AUTHORIZATION pg_database_owner;

-- Esquema Resources (Recursos)

CREATE SCHEMA Resources AUTHORIZATION pg_database_owner;
```

Las tablas se dividen así:

- *persons*
  - drivers
- *resources*
  - vehicles
  - routes

- trips
- deliveries
- maintenance

El diagrama ER quedaría así:



4. Instalar dependencias

Desde el PowerShell o desde el entorno que uses ejecutas el siguiente código

pip install -r requirements.txt

5. Configuración logging en data_generation.py

- Para empezar la generación de los datos debemos ingresar al motor de SQL

```
DB_CONFIG = {
    'host': 'localhost',
    'database': 'fleetlogix_database',
    'user': 'postgres',
    'password': 'password', # Aqui colocas tu contraseña
    'port': 5432
}
```

6. Explicación Código

- Usando POO creamos la primera clase DataGenerator donde se crean las funciones generadoras de Datos sintéticos mediante la librería Faker que instalamos e importamos anteriormente.
    - generate_vehicles()
      Esta funcion genera los datos de los vehículos y los va a dividir entre 4 tipos de vehículos y dos tipos de combustible. Ademas mediante un ciclo for va a crear los datos entre la lista de vehicle_types.
      Para crear las placas o licencias simplemente se crean letras y números aleatorios siguiendo un patron que es (ABC123) ya que asi son las placas en Colombia
    - generate_drivers()
      Esta empieza creando la lista de drivers y los tipos de licencia que en Colombia son A2, C1, C2, C3
      Luego se crean códigos de empleados y nombres usando la librería Faker con la función fake.first_name()
      Las licencias en Colombia tienen 10 digitos asi que creamos un numero entre 0000000000 y 999999999
    - generate_routes()

Para crear rutas reales y no tener el caso donde la ciudad de origen es la misma de destino o caso parecidos usamos un bucle anidado con un condicional donde tenemos: if origen != destination. Dentro de esta condición ya tenemos la generacion.
Algo importante a recalcar es la forma en la que se calculan los costos totales, la duración y la distancia en km. Para los costos obtenemos un valor promedio de peajes y tambien un valor aproximado del promedio de velocidad.

- _get_distance(origin, destination)
  Esta es la funcion que se usa en generate_routes() para conseguir las distancias entre ciudades
- generate_trips()
  Esta funcion crea los viajes que realizan los vehículos y los drivers y crea tambien otros valores para las demas columnas como por ejemplo el peso por viaje, el consumo de combustible, etc. Para hacer esto usamos random.uniform()
- _get_hourly_distribution()
  Se toman rangos de hora usando formato de 24 horas y se asignan probabilidades para crear los tiempos de entrega y de llegada
- generate_deliveries()
  Primero obtenemos los trips con su infromacion esto nos sirve para poder reutilizar esos datos y a partir de ahí crear los deliveries.
  Ya que los deliveries se crean entre los trips, usmaos un ciclo for para generar los datos.
  Para crear lo relacionado con el customer creamos el nombre, el numero de envio, la dirección y el peso del envio.
  Se crean los tiempos de entrega que deben ser mayores o iguales a los tiempos programados es decir o tienen que llegar a tiempo o con retraso.
- _distribute_weight()
  Esta funcion crea los pesos aleatorios y además los normaliza para no exceder el peso total y tambien se coloca la condición de que el peso mínimo es 0.5Kg
- generate_maintance()
  Eta funcion permite crear los mantenimientos partiendo de una consulta para obtener los datos de la tabla trips y de los vehículos.
  Ya que para crear los datos utilizan el primer y ultimo viaje usamos la función de ventana LAST_VALUES() y FIRST_VALUE().
  Se definen 6 tipos de mantenimiento.

Tambien se definen las fechas de mantenimiento para asi evitar que un mantenimiento este definido dentro de un dia que esta haciendo deliveries.

- validate_data_quality()
  Se verifica la integridad de los datos, en caso de tener errores o datos que no cumplan las condiciones en el archivo data_generation.log nos arrojara un error en la validación correspondiente
- generaty_summary_report()
  Esta funcion es para generar el reporte de la generacion de los datos en el archivo .log y guardar el resumen en un JSON.

7. Ejecutar codigo

Desde la terminal ejecutar el siguiente comando o desde vsc presionar Run Python File

python data_generation.py

- Se espera que se creen los siguientes archivos

data_generation.log

generation_summary.json