Carrera: Data Science

Módulo 1

FLEETLOGIX

Nombre del autor: Enzo A. Zambón

Email: enzo.a.zambon.16@gmail.com

Cohorte: PT01

Fecha de entrega: 08/10/2025

Introducción

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.

Recibimos una base de datos PostgreSQL con el modelo relacional básico ya implementado (6 tablas principales vacías). El desafio consiste en poblar todas las tablas con datos sintéticos masivos que respeten las relaciones del negocio, desarrollar queries para resolver problemas operativos reales, y diseñar una arquitectura cloud que permite análisis en tiempo real y toma de decisiones basada en datos.

Objetivos del PI:

- Dominar SQL con CTEs, Window Functions y optimización de queries
- Generar datos sintéticos masivos manteniendo integridad y realismo
- Transformar un modelo transaccional en un Data Warehouse dimensional
- Implementar arquitectura cloud básica con AWS (RDS, S3, Lambda, DynamoDB, API Gateway)
- Integrar bases NoSQL (MongoDB) para datos no estructurados
- Desarrollar pipeline ETL con Python
- Aplicar mejores prácticas de arquitectura de datos

Carrera: Data Science

Módulo 1

Desarrollo del proyecto

Avance 1: Análisis del modelo proporcionado

Documentación de Relaciones y Constraints:

- Tablas de Dimensiones:

1. Tabla: vehicles (vehículos de la flota)
    - Clave primaria (PK): vehicle_id. Es el identificador único de cada vehículo dentro de nuestra base
    - Constraints:
        - license_plate: UNIQUE NOT NULL (cada matrícula debe ser única y no nula).
        - vehicle_type: NOT NULL (es un campo obligatorio de relleno).
        - Status: por defecto todos los vehículos inician como activos ('active').

      Relaciones:

        - 1:N con trips (un vehículo puede realizar muchos viajes).
        - 1:N con maintenance (un vehículo puede tener múltiples mantenimientos).

2. Tabla: drivers (conductores)
    - Clave primaria (PK): driver_id. Es el identificador único de cada conductor.
    - Constraints:
        - employee_code: UNIQUE NOT NULL (código único de empleado).
        - license_number: UNIQUE NOT NULL (número de licencia único y no nulo).
        - first_name, last_name: las Casillas de nomre y apellido deben estar completas (NOT NULL)
        - status: por defecto 'active'.

      Relaciones:

        - 1:N con trips (un conductor puede realizar múltiples viajes).

3. Tabla: routes (rutas predefinidas):
   - Clave primaria (PK): route_id: Identificador único de la ruta.
   - Constraints:
     - route_code: UNIQUE NOT NULL (cada ruta tiene un código único).
     - origin_city, destination_city: Tiene que haber siempre un origen y un destino de los viajes. NOT NULL.
     - toll_cost: por defecto 0.

     Relaciones:

     - 1:N con trips (una ruta puede estar asociada a varios viajes).

➢ Tablas de hechos:

4. Tabla: trips (viajes realizados)
   - Clave primaria (PK): trip_id. Es el identificador único del viaje.
   - Constraints y Claves Foráneas (FK):
     - vehicle_id: FK que referencia a vehicles(vehicle_id).
     - driver_id: FK que referencia a drivers(driver_id).
     - route_id: FK que referencia a routes(route_id).
     - departure_datetime: NOT NULL (todo viaje debe tener salida).
     - Status: por defecto se le asigna un 'in_progress'.

     Relaciones:

     - N:1 con vehicles (cada viaje lo hace un vehículo).
     - N:1 con drivers (cada viaje lo realiza un conductor).
     - N:1 con routes (cada viaje sigue una ruta).
     - 1:N con deliveries (un viaje puede contener múltiples entregas).

5. Tabla: deliveries (entregas individuales):
   - Clave primaria (PK): delivery_id. Identificador único de la entrega.

- Constraints y Claves Foráneas (FK):
  - trip_id: FK que referencia a trips(trip_id).
  - tracking_number: UNIQUE NOT NULL (cada entrega tiene un número de seguimiento único).
  - customer_name, delivery_address: NOT NULL.
  - delivery_status; por defecto 'pending'.
  - recipient_signature: por defecto FALSE.

  Relaciones:

  - N:1 con trips (una entrega pertenece a un viaje).
  -

6. Tabla: maintenance (mantenimientos de vehículos)
   - Clave primaria (PK): maintenance_id. Es el Identificador único del mantenimiento.
   - Constraints y Claves Foráneas (FK):
     - vehicle_id: FK que referencia a vehicles(vehicle_id).
     - maintenance_date: NOT NULL.
     - maintenance_type: NOT NULL.

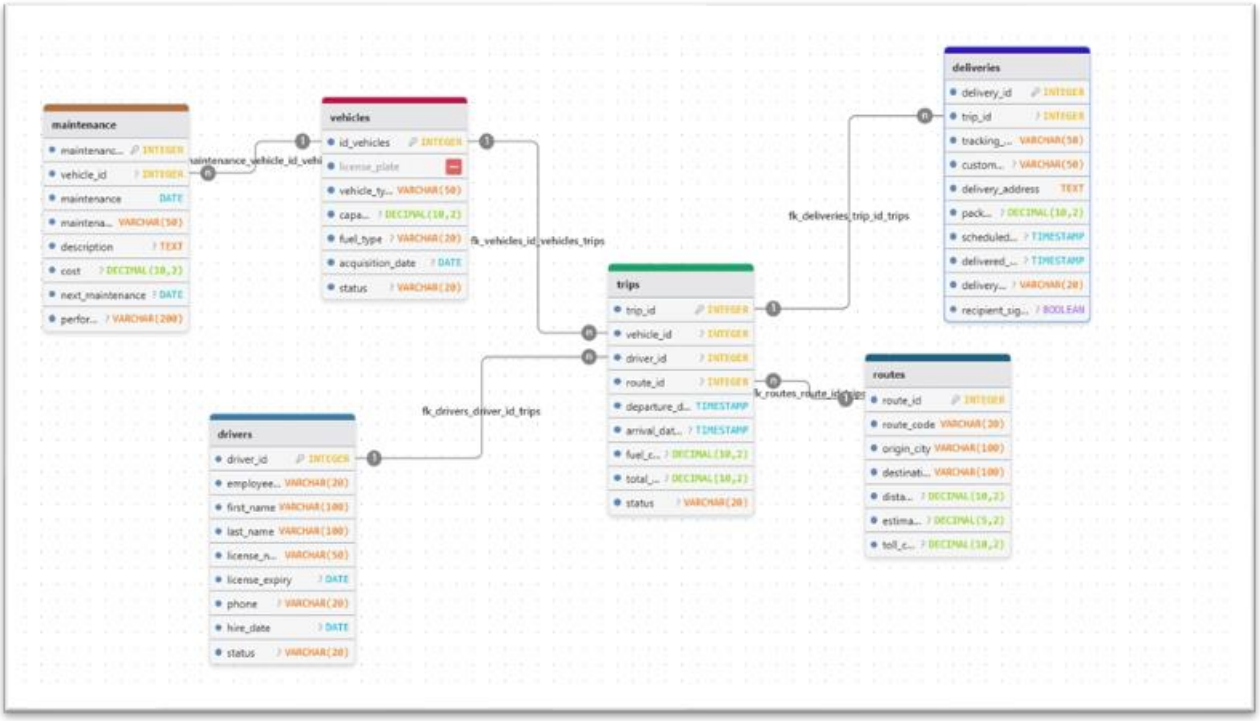
     Relaciones:

     - N:1 con vehicles (cada mantenimiento corresponde a un vehículo).

Diseño del esquema de la base de datos

Es útil pensar en cómo se vería nuestro esquema de relaciones dentro de la base de datos que vamos a estructurar. Para ello, diseñé un modelo relacional de acuerdo con la información suministrada por el Modelo Semántico en la capsula de Henry, identificando relaciones entre las tablas a partir de claves en común.

En dicho modelo fue posible identificar 3 tablas independientes o de dimensiones (aquellas que no dependen de otras tablas para su existencia, es decir, no tienen claves foráneas hacia otras tablas): Vehicles, Drivers y Routes. A su vez, se identifican también tres tablas dependientes o de hechos: Trips, Manteinance y Deliveris



Lo siguiente fue construir la base de datos en PostgreSQL, aplicando las siguientes sentencias:

1. Creación y selección de la base de datos a utilizar:

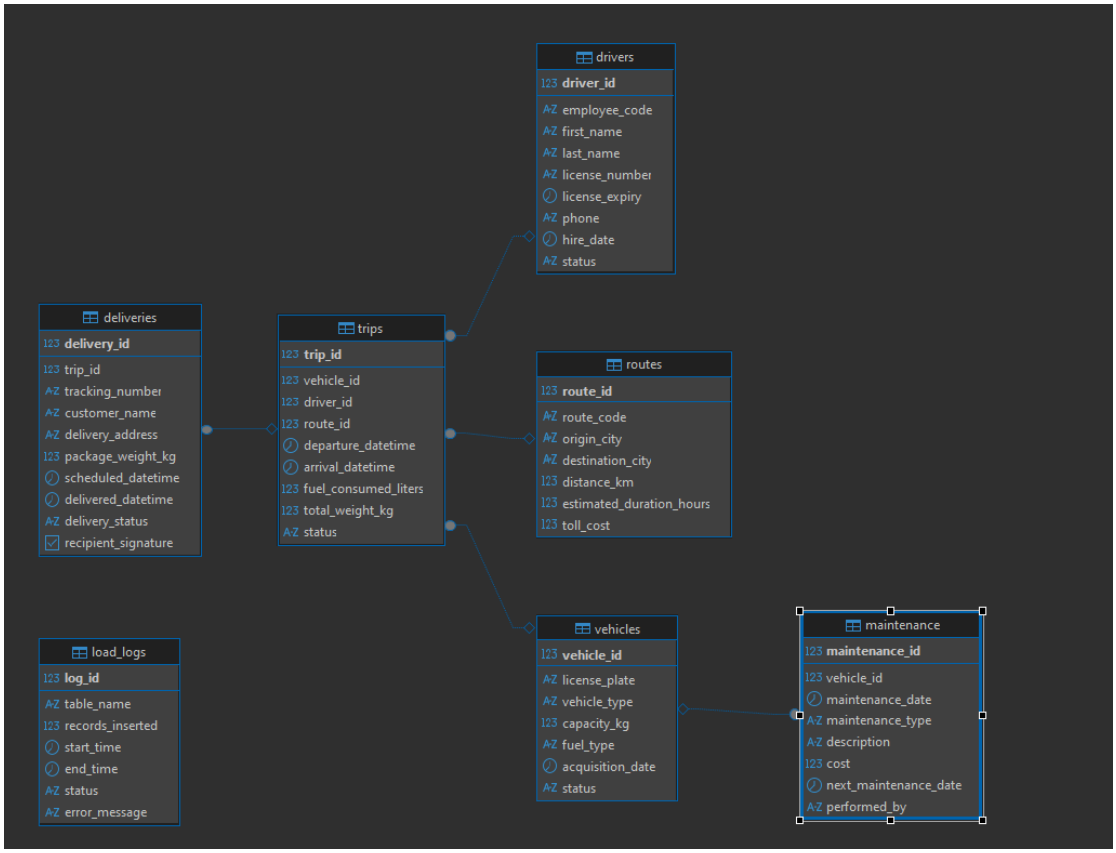
```
-- ============================================================
-- FLEETLOGIX DATABASE SETUP
-- Sistema de Gestión de Transporte y Logística
-- ============================================================

create database fleetlogix;
```

2. Creación de las tablas con sus respectivas columnas y relaciones (a partir del uso de claves foráneas):

```sql
-- 1. Creamos las tablas del modelo relacional

-- Tabla 1: vehicles (vehículos de la flota)
CREATE TABLE vehicles (
    vehicle_id SERIAL PRIMARY KEY,
    license_plate VARCHAR(20) UNIQUE NOT NULL,
    vehicle_type VARCHAR(50) NOT NULL,
    capacity_kg DECIMAL(10,2),
    fuel_type VARCHAR(20),
    acquisition_date DATE,
    status VARCHAR(20) DEFAULT 'active'
);

-- Tabla 2: drivers (conductores)
CREATE TABLE drivers (
    driver_id SERIAL PRIMARY KEY,
    employee_code VARCHAR(20) UNIQUE NOT NULL,
    first_name VARCHAR(100) NOT NULL,
    last_name VARCHAR(100) NOT NULL,
    license_number VARCHAR(50) UNIQUE NOT NULL,
    license_expiry DATE,
    phone VARCHAR(20),
    hire_date DATE,
    status VARCHAR(20) DEFAULT 'active'
);

-- Tabla 3: routes (rutas predefinidas)
CREATE TABLE routes (
    route_id SERIAL PRIMARY KEY,
    route_code VARCHAR(20) UNIQUE NOT NULL,
    origin_city VARCHAR(100) NOT NULL,
    destination_city VARCHAR(100) NOT NULL,
    distance_km DECIMAL(10,2),
    estimated_duration_hours DECIMAL(5,2),
    toll_cost DECIMAL(10,2) DEFAULT 0
);
```

```sql
-- Tabla 4: trips (viajes realizados)
CREATE TABLE trips (
    trip_id SERIAL PRIMARY KEY,
    vehicle_id INTEGER REFERENCES vehicles(vehicle_id),
    driver_id INTEGER REFERENCES drivers(driver_id),
    route_id INTEGER REFERENCES routes(route_id),
    departure_datetime TIMESTAMP NOT NULL,
    arrival_datetime TIMESTAMP,
    fuel_consumed_liters DECIMAL(10,2),
    total_weight_kg DECIMAL(10,2),
    status VARCHAR(20) DEFAULT 'in_progress'
);

-- Tabla 5: deliveries (entregas individuales)
CREATE TABLE deliveries (
    delivery_id SERIAL PRIMARY KEY,
    trip_id INTEGER REFERENCES trips(trip_id),
    tracking_number VARCHAR(50) UNIQUE NOT NULL,
    customer_name VARCHAR(200) NOT NULL,
    delivery_address TEXT NOT NULL,
    package_weight_kg DECIMAL(10,2),
    scheduled_datetime TIMESTAMP,
    delivered_datetime TIMESTAMP,
    delivery_status VARCHAR(20) DEFAULT 'pending',
    recipient_signature BOOLEAN DEFAULT FALSE
);

-- Tabla 6: maintenance (mantenimientos de vehículos)
CREATE TABLE maintenance (
    maintenance_id SERIAL PRIMARY KEY,
    vehicle_id INTEGER REFERENCES vehicles(vehicle_id),
    maintenance_date DATE NOT NULL,
    maintenance_type VARCHAR(50) NOT NULL,
    description TEXT,
    cost DECIMAL(10,2),
    next_maintenance_date DATE,
    performed_by VARCHAR(200)
);
```

Y nos queda el siguiente esquema relacional:



Tambien creamos los índices y agregamos los comentarios correspondientes:

```sql
-- 2. Crear índices básicos proporcionados
CREATE INDEX idx_trips_departure ON trips(departure_datetime);
CREATE INDEX idx_deliveries_status ON deliveries(delivery_status);
CREATE INDEX idx_vehicles_status ON vehicles(status);

-- 3. Agregar comentarios a las tablas para documentación
COMMENT ON TABLE vehicles IS 'Registro de vehículos de la flota de FleetLogix';
COMMENT ON TABLE drivers IS 'Información de conductores empleados';
COMMENT ON TABLE routes IS 'Rutas predefinidas entre ciudades';
COMMENT ON TABLE trips IS 'Registro de viajes realizados';
COMMENT ON TABLE deliveries IS 'Entregas individuales asociadas a cada viaje';
COMMENT ON TABLE maintenance IS 'Historial de mantenimiento de vehículos';
```

Verificamos la creación de las tablas:

```sql
-- 4. Verificar la creación de las tablas
SELECT
    table_name,
    (SELECT COUNT(*) FROM information_schema.columns
     WHERE table_schema = 'public'
     AND table_name = t.table_name) as column_count
FROM information_schema.tables t
WHERE table_schema = 'public'
AND table_type = 'BASE TABLE'
ORDER BY table_name;
```

tables 1

SELECT table_name, (SELECT COUNT(*) FROM info | Enter a SQL expression to filter results (

Grilla | Texto

	table_name	column_count
1	deliveries	10
2	drivers	9
3	maintenance	8
4	routes	7
5	trips	9
6	vehicles	7

Y las relaciones:

```sql
-- 5. Verificar las relaciones (foreign keys)
SELECT
    tc.table_name AS tabla_origen,
    kcu.column_name AS columna_origen,
    ccu.table_name AS tabla_referencia,
    ccu.column_name AS columna_referencia
FROM information_schema.table_constraints AS tc
JOIN information_schema.key_column_usage AS kcu
    ON tc.constraint_name = kcu.constraint_name
    AND tc.table_schema = kcu.table_schema
JOIN information_schema.constraint_column_usage AS ccu
    ON ccu.constraint_name = tc.constraint_name
    AND ccu.table_schema = tc.table_schema
WHERE tc.constraint_type = 'FOREIGN KEY'
AND tc.table_schema = 'public';
```

table_constraints(+) 1

SELECT tc.table_name AS tabla_origen, kcu.column | Enter a SQL expression to filter results (use Ctrl+Space)

	tabla_origen	columna_origen	tabla_referencia	columna_referencia
1	trips	vehicle_id	vehicles	vehicle_id
2	trips	driver_id	drivers	driver_id
3	trips	route_id	routes	route_id
4	deliveries	trip_id	trips	trip_id
5	maintenance	vehicle_id	vehicles	vehicle_id

Por último, verificamos los índices creados:

```sql
-- 6. Verificar índices creados
SELECT
    schemaname,
    tablename,
    indexname,
    indexdef
FROM pg_indexes
WHERE schemaname = 'public'
ORDER BY tablename, indexname;
```

pg_indexes 1

Enter a SQL expression to filter results (use Ctrl+Space)

	schemaname	tablename	indexname	indexdef
1	public	deliveries	deliveries_pkey	CREATE UNIQUE INDEX deliveries_pkey ON public.deliveries US
2	public	deliveries	deliveries_tracking_number_key	CREATE UNIQUE INDEX deliveries_tracking_number_key ON pub
3	public	deliveries	idx_deliveries_status	CREATE INDEX idx_deliveries_status ON public.deliveries USING
4	public	drivers	drivers_employee_code_key	CREATE UNIQUE INDEX drivers_employee_code_key ON public.
5	public	drivers	drivers_license_number_key	CREATE UNIQUE INDEX drivers_license_number_key ON public.c
6	public	drivers	drivers_pkey	CREATE UNIQUE INDEX drivers_pkey ON public.drivers USING b
7	public	maintenance	maintenance_pkey	CREATE UNIQUE INDEX maintenance_pkey ON public.maintena

Renovar | Save | Cancel | Exportar datos ... | 200 | 14 | 14 row(s) fetched

Finalmente, creamos una tabla de Logs para que se registren nuestras ingestas

```sql
--- CREAR TABLA DE LOGS

CREATE TABLE IF NOT EXISTS load_logs (
    log_id SERIAL PRIMARY KEY,
    table_name VARCHAR(50) NOT NULL,
    records_inserted INTEGER,
    start_time TIMESTAMP DEFAULT now(),
    end_time TIMESTAMP,
    status VARCHAR(20),
    error_message TEXT
);
```

3. El siguiente paso consistió en llenar las tablas previamente creadas con datos y gestionar dicha información utilizando Phyton.

   Primero importamos las librerías a utilizar y creamos la conexión a PostgreSQL para poder realizar la injerta de datos.

   - Psycopg2: Es para conectarse a bases de datos PostgreSQL

- Faker: Se usa para generar datos falsos pero realistas. En el código actual, genera vehículos, conductores, rutas, entregas y mantenimiento de manera automática y aleatoria.
- Datetime: Librería estándar de Python para manejar fechas y horas. En este código se usa para definir fechas de adquisición, contratación, licencias, viajes y entregas y calcular tiempos de salida y llegada de viajes.
- RANDOM_SEED = 42. Define una semilla fija para los generadores aleatorios. Usar siempre la misma semilla hace que los resultados se repitan (muy útil para pruebas y depuración).
- random.seed(RANDOM_SEED). Fija la semilla del módulo estándar de Python random.
- np.random.seed(RANDOM_SEED). Fija la semilla del generador aleatorio de NumPy.

```python
import psycopg2
from faker import Faker
import random
import pandas as pd
import numpy as np
from datetime import datetime, timedelta

# ---------- Configuración Semilla ----------
fake = Faker("es_ES")
RANDOM_SEED = 42
random.seed(RANDOM_SEED)
np.random.seed(RANDOM_SEED)

# ---------- Conexión a PostgreSQL ----------
conn = psycopg2.connect(
    dbname="fleetlogix",
    user="postgres",
    password="Spriest123",
    host="localhost",
    port="5432"
)
cur = conn.cursor()
```

Luego, creamos una función para registrar los logs

```
# ---------- # Función para registrar logs ----------
def log_load(table_name, records_inserted, status, error_message=None):
    cur_log = conn.cursor()
    cur_log.execute("""
        INSERT INTO load_logs (table_name, records_inserted, end_time, status, error_message)
        VALUES (%s,%s,%s,%s,%s)
    """, (table_name, records_inserted, datetime.now(), status, error_message))
    conn.commit()
    cur_log.close()
```

Lo siguiente es poblar las tablas de dimensiones:

```
# ---------- 1 Poblar vehicles (200) ----------
vehicle_types = ["Camión grande", "Camión mediano", "Van", "Motocicleta"]
fuel_types = ["Diesel", "Nafta", "Eléctrico"]
vehicle_capacity_ranges = {
    "Camión grande": (12000, 20000),
    "Camión mediano": (6000, 12000),
    "Van": (1000, 4000),
    "Motocicleta": (50, 200),
}

def sample_capacity_for_type(vtype):
    low, high = vehicle_capacity_ranges[vtype]
    return random.randint(low, high)

vehicles_data = []
for _ in range(200):
    vt = random.choice(vehicle_types)
    capacity_kg = sample_capacity_for_type(vt)
    vehicles_data.append((
        fake.unique.license_plate(),
        vt,
        capacity_kg,
        random.choice(fuel_types),
        fake.date_between(start_date="-10y", end_date="today"),
        random.choice(["active", "inactive", "maintenance"])
    ))

cur.executemany("""
INSERT INTO vehicles (license_plate, vehicle_type, capacity_kg, fuel_type, acquisition_date, status)
VALUES (%s,%s,%s,%s,%s,%s)
""", vehicles_data)
conn.commit()
log_load("vehicles", len(vehicles_data), "success")
print("✅ Vehicles insertados")

vehicle_id_to_type = {i+1: vehicles_data[i][1] for i in range(len(vehicles_data))}
vehicle_id_to_capacity = {i+1: vehicles_data[i][2] for i in range(len(vehicles_data))}
vehicle_id_to_status = {i+1: vehicles_data[i][5] for i in range(len(vehicles_data))}
```

```python
# ---------- 2 Poblar drivers (400) ----------
drivers_data = []
for _ in range(400):
    drivers_data.append((
        fake.unique.bothify(text="EMP###"),
        fake.first_name(),
        fake.last_name(),
        fake.unique.bothify(text="LIC#####"),
        fake.date_between(start_date="today", end_date="+1y"),
        fake.phone_number(),
        fake.date_between(start_date="-10y", end_date="today"),
        random.choice(["active", "inactive"])
    ))

cur.executemany("""
INSERT INTO drivers (employee_code, first_name, last_name, license_number, license_expiry, phone, hire_date, status)
VALUES (%s,%s,%s,%s,%s,%s,%s,%s)
""", drivers_data)
conn.commit()
log_load("drivers", len(drivers_data), "success")
print("✅ Drivers insertados")

driver_id_to_status = {i+1: drivers_data[i][7] for i in range(len(drivers_data))}
```

```python
# ---------- 3 Poblar routes (50) ----------
cities = ["Buenos Aires", "Rosario", "Córdoba", "Mendoza", "La Plata", "Salta", "Mar del Plata", "Bahía Blanca"]
routes_data = []
for i in range(50):
    origin, destination = random.sample(cities, 2)
    distance = round(random.uniform(20, 1200), 2)
    est_duration = round(max(0.5, distance / random.uniform(40, 90)), 2)
    toll_cost = round(random.uniform(0, 5000), 2)
    routes_data.append((f"R{i+1:03d}", origin, destination, distance, est_duration, toll_cost))

cur.executemany("""
INSERT INTO routes (route_code, origin_city, destination_city, distance_km, estimated_duration_hours, toll_cost)
VALUES (%s,%s,%s,%s,%s,%s)
""", routes_data)
conn.commit()
log_load("routes", len(routes_data), "success")
print("✅ Routes insertadas")

routes_df = pd.DataFrame({
    'route_id': list(range(1, len(routes_data) + 1)),
    'route_code': [r[0] for r in routes_data],
    'origin_city': [r[1] for r in routes_data],
    'destination_city': [r[2] for r in routes_data],
    'distance_km': [r[3] for r in routes_data],
    'estimated_duration_hours': [r[4] for r in routes_data]
})
```

A continuación, para poder insertar datos en la tabla de viajes, lo primero que hay que hacer es generarlos y para ello, utilizamos una función auxiliar gethourly_distribution, la cual genera un vector de 24 probabilidades, una para cada hora del día (0–23). Estas probabilidades se usan para decidir a qué hora ocurren los viajes (departure_datetime) de manera realista. Los parámetros de la función son:

A. peacks: Lista de horas pico (por ejemplo [8, 9, 17] para la mañana y tarde).
B. peak_weights: Cuán probable es que ocurra un evento en cada hora pico. Si no se pasa nada, todas tienen peso 3.
C. Spread: permite “difuminar” el pico hacia las horas cercanas, haciendo la distribución más suave.

```python
# Función auxiliar: distribución horaria
def get_hourly_distribution(peaks=None, peak_weights=None, spread=1):
    w = np.ones(24, dtype=float)
    if peaks:
        if peak_weights is None:
            peak_weights = [3] * len(peaks)
        for h, wt in zip(peaks, peak_weights):
            w[int(h) % 24] += wt
    if spread > 0:
        kernel = np.ones(2 * spread + 1, dtype=float)
        w = np.convolve(w, kernel, mode='same')
    return w / w.sum()
```

Es útil en el código porque se asegura que los viajes no sean distribuidos de forma completamente uniforme durante el día. De esta manera, simula comportamientos reales: más viajes en horas pico, menos en la madrugada.

Podemos también validar la consistencia temporal entre salida y llegada mediante la siguiente función

```python
def ensure_arrival_after_departure(dep_dt, arr_dt):
    if arr_dt is None:
        return None
    if pd.isna(arr_dt):
        return None
    if arr_dt <= dep_dt:
        return dep_dt + timedelta(minutes=5)
    if (arr_dt - dep_dt) < timedelta(minutes=5):
        return dep_dt + timedelta(minutes=5)
    return arr_dt
```

La siguiente función es para simular viajes de la flota con horarios realistas; distribución aleatoria de vehículos, conductores y rutas; Combustible y peso realistas y estado del viaje.

```python
# Función principal: generar trips
def generate_trips(n_trips, start_date, end_date, vehicle_ids, driver_ids, routes_df, hourly_dist=None, seed=None):
    if seed is not None:
        random.seed(seed)
        np.random.seed(seed)

    start = pd.to_datetime(start_date)
    end = pd.to_datetime(end_date)
    total_days = max(1, (end - start).days)

    vehicle_ids_active = [vid for vid in vehicle_ids if vehicle_id_to_status.get(vid) == "active"]
    if not vehicle_ids_active:
        vehicle_ids_active = vehicle_ids[:]
    driver_ids_active = [did for did in driver_ids if driver_id_to_status.get(did) == "active"]
    if not driver_ids_active:
        driver_ids_active = driver_ids[:]

    chosen_vehicles = np.random.choice(vehicle_ids_active, size=n_trips, replace=True)
    chosen_drivers = np.random.choice(driver_ids_active, size=n_trips, replace=True)
    chosen_routes_idx = np.random.choice(routes_df.index.to_numpy(), size=n_trips, replace=True)
    chosen_routes = routes_df.loc[chosen_routes_idx].reset_index(drop=True)

    hourly_probs = get_hourly_distribution() if hourly_dist is None else np.array(hourly_dist) / np.sum(hourly_dist)

    day_offsets = np.random.randint(0, total_days, size=n_trips)
    chosen_hours = np.random.choice(np.arange(24), size=n_trips, p=hourly_probs)
    chosen_minutes = np.random.randint(0, 60, size=n_trips)
    chosen_seconds = np.random.randint(0, 60, size=n_trips)

    dep_dates = pd.to_datetime(start) + pd.to_timedelta(day_offsets, unit='d') \
              + pd.to_timedelta(chosen_hours, unit='h') \
              + pd.to_timedelta(chosen_minutes, unit='m') \
              + pd.to_timedelta(chosen_seconds, unit='s')

    durations = chosen_routes['estimated_duration_hours'].to_numpy()
    noise_hours = np.random.uniform(-0.25, 1.0, size=n_trips)

    arrival_proposals = dep_dates + pd.to_timedelta(np.maximum(0.1, durations + noise_hours), unit='h')
    distances = chosen_routes['distance_km'].to_numpy()
```

```python
    total_weight = []
    fuel_consumed = []
    final_arrivals = []
    for i, vid in enumerate(chosen_vehicles):
        cap = vehicle_id_to_capacity.get(int(vid), 1000)
        v_status = vehicle_id_to_status.get(int(vid), "active")
        if v_status == "active":
            w = round(random.uniform(50, cap), 2)
        else:
            w = round(random.uniform(50, min(500, cap)), 2)
        total_weight.append(w)
        vtype = vehicle_id_to_type.get(int(vid), "Van")
        fuel_consumed.append(round(distances[i] * sample_fuel_rate_for_type(vtype), 2))

        arr_prop = arrival_proposals[i].to_pydatetime() if not pd.isna(arrival_proposals[i]) else None
        dep_dt = dep_dates[i].to_pydatetime()
        final_arrivals.append(ensure_arrival_after_departure(dep_dt, arr_prop))

    statuses = np.random.choice(['completed', 'in_progress', 'cancelled'], size=n_trips, p=[0.85, 0.10, 0.05])

    df = pd.DataFrame({
        'vehicle_id': chosen_vehicles.astype(int),
        'driver_id': chosen_drivers.astype(int),
        'route_id': chosen_routes['route_id'].to_numpy().astype(int),
        'departure_datetime': dep_dates,
        'arrival_datetime': final_arrivals,
        'fuel_consumed_liters': np.round(fuel_consumed, 2),
        'total_weight_kg': np.round(total_weight, 2),
        'distance_km': distances,
        'estimated_duration_hours': durations,
        'status': statuses
    })

    cancelled_mask = df['status'] == 'cancelled'
    df.loc[cancelled_mask, 'arrival_datetime'] = None
    df.loc[cancelled_mask, 'total_weight_kg'] = 0.0
    df.loc[cancelled_mask, 'fuel_consumed_liters'] = 0.0

    for idx, row in df.loc[~cancelled_mask].iterrows():
        dep = pd.to_datetime(row['departure_datetime'])
        arr = row['arrival_datetime']
        if arr is None or pd.isna(arr):
            df.at[idx, 'arrival_datetime'] = dep + timedelta(minutes=5)
        else:
            arr = pd.to_datetime(arr)
            if arr <= dep + timedelta(seconds=0):
                df.at[idx, 'arrival_datetime'] = dep + timedelta(minutes=5)
            elif (arr - dep) < timedelta(minutes=1):
                df.at[idx, 'arrival_datetime'] = dep + timedelta(minutes=5)

    df.reset_index(drop=True, inplace=True)
    df.insert(0, 'trip_id', np.arange(1, len(df) + 1))
    return df
```

```
total_weight = []
fuel_consumed = []
final_arrivals = []
for i, vid in enumerate(chosen_vehicles):
    cap = vehicle_id_to_capacity.get(int(vid), 1000)
    v_status = vehicle_id_to_status.get(int(vid), "active")
    if v_status == "active":
        w = round(random.uniform(50, cap), 2)
    else:
        w = round(random.uniform(50, min(500, cap)), 2)
    total_weight.append(w)
    vtype = vehicle_id_to_type.get(int(vid), "Van")
    fuel_consumed.append(round(distances[i] * sample_fuel_rate_for_type(vtype), 2))

    arr_prop = arrival_proposals[i].to_pydatetime() if not pd.isna(arrival_proposals[i]) else None
    dep_dt = dep_dates[i].to_pydatetime()
    final_arrivals.append(ensure_arrival_after_departure(dep_dt, arr_prop))

statuses = np.random.choice(['completed', 'in_progress', 'cancelled'], size=n_trips, p=[0.85, 0.10, 0.05])

df = pd.DataFrame({
    'vehicle_id': chosen_vehicles.astype(int),
    'driver_id': chosen_drivers.astype(int),
    'route_id': chosen_routes['route_id'].to_numpy().astype(int),
    'departure_datetime': dep_dates,
    'arrival_datetime': final_arrivals,
    'fuel_consumed_liters': np.round(fuel_consumed, 2),
    'total_weight_kg': np.round(total_weight, 2),
    'distance_km': distances,
    'estimated_duration_hours': durations,
    'status': statuses
})

cancelled_mask = df['status'] == 'cancelled'
df.loc[cancelled_mask, 'arrival_datetime'] = None
df.loc[cancelled_mask, 'total_weight_kg'] = 0.0
df.loc[cancelled_mask, 'fuel_consumed_liters'] = 0.0

for idx, row in df.loc[~cancelled_mask].iterrows():
    dep = pd.to_datetime(row['departure_datetime'])
    arr = row['arrival_datetime']
    if arr is None or pd.isna(arr):
        df.at[idx, 'arrival_datetime'] = dep + timedelta(minutes=5)
    else:
        arr = pd.to_datetime(arr)
        if arr <= dep + timedelta(seconds=0):
            df.at[idx, 'arrival_datetime'] = dep + timedelta(minutes=5)
        elif (arr - dep) < timedelta(minutes=1):
            df.at[idx, 'arrival_datetime'] = dep + timedelta(minutes=5)

df.reset_index(drop=True, inplace=True)
df.insert(0, 'trip_id', np.arange(1, len(df) + 1))
return df
```

Poblamos la tabla de viajes:

```python
# Generar trips
hourly_dist = get_hourly_distribution(peaks=[8, 9, 17], peak_weights=[4, 3, 5], spread=1)
df_trips = generate_trips(
    n_trips=100000,
    start_date='2023-01-01',
    end_date=datetime.today().strftime('%Y-%m-%d'),
    vehicle_ids=list(range(1, len(vehicles_data) + 1)),
    driver_ids=list(range(1, len(drivers_data) + 1)),
    routes_df=routes_df,
    hourly_dist=hourly_dist,
    seed=RANDOM_SEED
)


def row_to_tuple_safe(row):
    trip_id = int(row['trip_id'])
    vehicle_id = int(row['vehicle_id'])
    driver_id = int(row['driver_id'])
    route_id = int(row['route_id'])
    dep = row['departure_datetime'].to_pydatetime() if not pd.isna(row['departure_datetime']) else None
    arr = row['arrival_datetime']
    if pd.isna(arr):
        arr = None
    fuel = float(row['fuel_consumed_liters']) if not pd.isna(row['fuel_consumed_liters']) else 0.0
    weight = float(row['total_weight_kg']) if not pd.isna(row['total_weight_kg']) else 0.0
    status = row['status']
    return (trip_id, vehicle_id, driver_id, route_id, dep, arr, fuel, weight, status)

trip_tuples = [row_to_tuple_safe(row) for _, row in df_trips.iterrows()]


BATCH = 5000
for i in range(0, len(trip_tuples), BATCH):
    chunk = trip_tuples[i:i+BATCH]
    cur.executemany("""
    INSERT INTO trips (trip_id, vehicle_id, driver_id, route_id, departure_datetime, arrival_datetime, fuel_consumed_liters, total_weight_kg, status)
    VALUES (%s,%s,%s,%s,%s,%s,%s,%s,%s)
    """, chunk)
    conn.commit()

log_load("trips", len(trip_tuples), "success")
print("✅ Trips insertados:", len(trip_tuples))
```

Y las tablas restantes:

```python
# ---------- 5) Poblar deliveries (400,000) ----------
print("Generando deliveries...")
delivery_data = []
tracking_counter = 0
for idx, row in df_trips.iterrows():
    trip_id = int(row['trip_id'])
    if row['status'] == 'cancelled':
        continue
    num_deliveries = int(np.random.choice([2,3,4,5,6], p=[0.1,0.2,0.4,0.2,0.1]))
    total_weight = float(row['total_weight_kg']) if row['total_weight_kg'] > 0 else round(random.uniform(1,50), 2)
    portions = np.random.dirichlet(alpha=np.ones(num_deliveries))
    package_weights = np.round(portions * total_weight, 2)
    departure = row['departure_datetime'].to_pydatetime() if not pd.isna(row['departure_datetime']) else None
    arrival = row['arrival_datetime'].to_pydatetime() if (row['arrival_datetime'] is not None and not pd.isna(row['arrival_datetime'])) else None
    base_times = []
    if arrival is not None:
        for i in range(num_deliveries):
            base_times.append(departure + (arrival - departure) * (i + 1) / (num_deliveries + 1))
    else:
        for i in range(num_deliveries):
            base_times.append(departure + timedelta(minutes=10*(i+1)))

    for j in range(num_deliveries):
        scheduled = base_times[j]
        delivered_dt = scheduled + timedelta(minutes=random.randint(0,30))
        if random.random() > 0.1:
            status = "delivered"
            signature = random.choice([True, False])
            delivered_value = delivered_dt
        else:
            status = random.choice(["pending", "cancelled"])
            signature = False
            delivered_value = None
        tracking_counter += 1
        tracking_number = f"TRK{trip_id:06d}{j+1:02d}{random.randint(1000,9999)}"
        delivery_data.append((
            trip_id,
            tracking_number,
            fake.name(),
            fake.address().replace("\n", " , "),
            float(package_weights[j]),
            scheduled,
            delivered_value,
            status,
            signature
        ))
    if len(delivery_data) >= 400000:
        break

# Insert deliveries in batches
BATCH = 5000
for i in range(0, len(delivery_data), BATCH):
    chunk = delivery_data[i:i+BATCH]
    cur.executemany("""
    INSERT INTO deliveries (trip_id, tracking_number, customer_name, delivery_address, package_weight_kg, scheduled_datetime, delivered_datetime, delivery_status, recipient_signature)
    VALUES (%s,%s,%s,%s,%s,%s,%s,%s,%s)
    """, chunk)
    conn.commit()

log_load("deliveries", len(delivery_data), "success")
print("✅ Deliveries insertadas:", len(delivery_data))
```

```python
# ---------- 6 Poblar maintenance (5,000) ----------
maintenance_types = [
    "Cambio de aceite","Revisión de frenos","Cambio de llantas",
    "Mantenimiento general","Revisión de motor","Alineación y balanceo"
]
maintenance_descriptions = {
    "Cambio de aceite": ["Se reemplazó el aceite del motor y se verificó el filtro.", "Cambio de aceite con filtro original y control de niveles."],
    "Revisión de frenos": ["Inspección de discos y pastillas; reemplazo si necesario.", "Ajuste y purga de líquido de frenos."],
    "Cambio de llantas": ["Sustitución por desgaste según índice de profundidad.", "Cambio de llanta pinchada y balanceo."],
    "Mantenimiento general": ["Servicio preventivo: filtros, niveles y revisión visual.", "Chequeo integral por kilometraje programado."],
    "Revisión de motor": ["Diagnóstico de inyección, bujías y correas.", "Inspección por consumo de aceite y fugas."],
    "Alineación y balanceo": ["Alineación de dirección y balanceo por vibraciones detectadas.", "Alineación tras cambio de suspensión."]
}
maintenance_cost_ranges = {
    "Cambio de aceite": (5000,10000),
    "Revisión de frenos": (8000,15000),
    "Cambio de llantas": (20000,40000),
    "Mantenimiento general": (15000,30000),
    "Revisión de motor": (30000,50000),
    "Alineación y balanceo": (7000,12000)
}
maintenance_data = []
for _ in range(5000):
    vehicle_id = random.randint(1, len(vehicles_data))
    mtype = random.choice(maintenance_types)
    desc = random.choice(maintenance_descriptions[mtype])
    low, high = maintenance_cost_ranges[mtype]
    cost = round(random.uniform(low, high), 2)
    maintenance_date = fake.date_between(start_date="-2y", end_date="today")
    next_date = maintenance_date + timedelta(days=random.randint(90, 365))
    maintenance_data.append((vehicle_id, maintenance_date, mtype, desc, cost, next_date, fake.company()))

cur.executemany("""
INSERT INTO maintenance (vehicle_id, maintenance_date, maintenance_type, description, cost, next_maintenance_date, performed_by)
VALUES (%s,%s,%s,%s,%s,%s,%s)
""", maintenance_data)
conn.commit()
log_load("maintenance", len(maintenance_data), "success")
print("✅ Maintenance insertados")

# ---------- Commit y cierre ----------
cur.close()
conn.close()
print("🚀 CARGA MASIVA COMPLETADA EXITOSAMENTE!")
print(f"✅ vehicles: {len(vehicles_data)}, drivers: {len(drivers_data)}, routes: {len(routes_data)}, trips: {len(trip_tuples)}, deliveries: {len(delivery_data)}, maintenance: {len(maintenance_data)}")
```

Para validar la integridad referencial completa de las tablas, realizamos consultas diversas:

a. Contar la cantidad de registros por tabla
b. Ver los primeros 5 registros de cada tabla
c. Contar cuantos vehículos/viajes/mantenimientos se encuentran en cada status
d. Contar los vehículos por tipo de combustible
e. Contar el número de viajes por tipo de vehículo
f. Seleccionar un conductor y ver la cantidad de viajes que hizo

```sql
-- Contar los registros totales de cada tabla --
SELECT 'vehicles' AS table_name, COUNT(*) AS total FROM vehicles
UNION ALL
SELECT 'drivers', COUNT(*) FROM drivers
UNION ALL
SELECT 'routes', COUNT(*) FROM routes
UNION ALL
SELECT 'trips', COUNT(*) FROM trips
UNION ALL
SELECT 'deliveries', COUNT(*) FROM deliveries
UNION ALL
SELECT 'maintenance', COUNT(*) FROM maintenance;


-- Podemos ver los primeros registros de cada tabla --
SELECT * FROM vehicles LIMIT 5;
SELECT * FROM drivers LIMIT 5;
SELECT * FROM routes LIMIT 5;
SELECT * FROM trips LIMIT 5;
SELECT * FROM deliveries LIMIT 5;
```

Resultados 1

Enter a SQL expression to filter results (use Ctrl+Space)

	table_name	total
1	vehicles	200
2	drivers	400
3	routes	50
4	trips	100.000
5	deliveries	379.553
6	maintenance	5.000

Podemos ver los primeros registros de cada tabla

```sql
-- Podemos ver los primeros registros de cada tabla --
SELECT * FROM vehicles LIMIT 5;
SELECT * FROM drivers LIMIT 5;
SELECT * FROM routes LIMIT 5;
SELECT * FROM trips LIMIT 5;
SELECT * FROM deliveries LIMIT 5;
SELECT * FROM maintenance LIMIT 5;
```

vehicles 1

Enter a SQL expression to filter results (use Ctrl+Space)

	vehicle_id	license_plate	vehicle_type	capacity_kg	fuel_type	acquisition_date	status
1	1	GC 6174 AI	Camión grande	12.204	Eléctrico	2017-08-16	inactive
2	2	9198 NHR	Camión mediano	7.828	Diesel	2024-05-21	maintenance
3	3	4323 STY	Camión grande	17.543	Eléctrico	2021-09-01	maintenance
4	4	H 7776 OF	Camión grande	16.837	Nafta	2018-07-12	active
5	5	1205 YNR	Camión grande	12.767	Diesel	2023-01-23	active

Contar cuantos vehículos, viajes, deliveries, se encuentran en cada estado:

```sql
-- Contar cuantos vehículos / viajes / deliveries se encuentran en cada estado --
SELECT status, COUNT(*) FROM vehicles GROUP BY status;
SELECT status, COUNT(*) FROM trips GROUP BY status;
SELECT delivery_status, COUNT(*) FROM deliveries GROUP BY delivery_status;
```

hicles 1

ELECT status, COUNT(*) FROM vehicles GROUP BY | Enter a SQL expression to filter results (use Ctrl+Space)

	A-Z status	123 count
1	inactive	71
2	active	63
3	maintenance	66

Contar los vehículos por tipo de combustible:

```sql
-- Contar los vehiculos por tipo de combustible --
SELECT fuel_type, COUNT(*) AS num_vehicles
FROM vehicles
GROUP BY fuel_type;
```

vehicles 1

SELECT fuel_type, COUNT(*) AS num_vehicles FROM | Enter a SQL expression to filter

	A-Z fuel_type	123 num_vehicles
1	Nafta	71
2	Diesel	69
3	Eléctrico	60

Seleccionar un conductor y ver la cantidad de viajes que realizó:

```sql
--- Seleccionar un conductor y ver la cantidad de viajes que hizo ---
SELECT driver_id,
       employee_code,
       first_name,
       last_name,
       license_number,
       license_expiry,
       phone,
       hire_date,
       status
FROM drivers
ORDER BY driver_id;


SELECT d.first_name || ' ' || d.last_name AS driver_name,
       COUNT(t.trip_id) AS num_trips
FROM trips t
JOIN drivers d ON t.driver_id = d.driver_id
WHERE d.first_name = 'Nidia' AND d.last_name = 'Quintana'
GROUP BY d.first_name, d.last_name;
```

Resultados 1

Enter a SQL expression to filter results (use Ctrl+Space)

	driver_name	num_trips
1	Nidia Quintana	545

Asegurar la consistencia temporal:

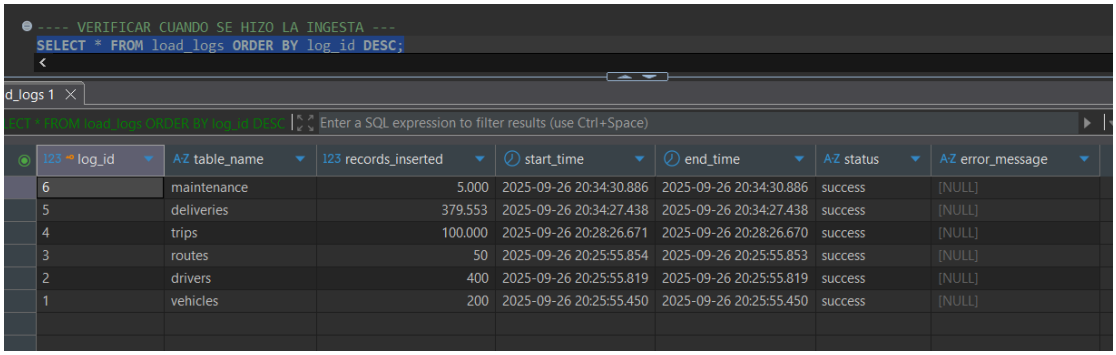
La siguiente consulta devuelve todos los viajes donde la llegada es antes o igual a la salida. Si no devuelve filas entonces todos los viajes son consistentes.

```sql
--- ASEGURAR LA CONSISTENCIA TEMPORAL ---
SELECT trip_id, departure_datetime, arrival_datetime
FROM trips
WHERE arrival_datetime <= departure_datetime;
```

Enter a SQL expression to filter results (use Ctrl+Space)

trip_id	departure_datetime	arrival_datetime

Por último, verificamos cuando se hizo la ingesta de cada tabla



```
---- VERIFICAR CUANDO SE HIZO LA INGESTA ---
SELECT * FROM load_logs ORDER BY log_id DESC;
```

d_logs 1

ECT * FROM load_logs ORDER BY log_id DESC | Enter a SQL expression to filter results (use Ctrl+Space)

log_id	table_name	records_inserted	start_time	end_time	status	error_message
6	maintenance	5.000	2025-09-26 20:34:30.886	2025-09-26 20:34:30.886	success	[NULL]
5	deliveries	379.553	2025-09-26 20:34:27.438	2025-09-26 20:34:27.438	success	[NULL]
4	trips	100.000	2025-09-26 20:28:26.671	2025-09-26 20:28:26.670	success	[NULL]
3	routes	50	2025-09-26 20:25:55.854	2025-09-26 20:25:55.853	success	[NULL]
2	drivers	400	2025-09-26 20:25:55.819	2025-09-26 20:25:55.819	success	[NULL]
1	vehicles	200	2025-09-26 20:25:55.450	2025-09-26 20:25:55.450	success	[NULL]

Avance 2: Ejecución y análisis de 12 queries proporcionadas

Una vez poblada la base, la empresa busca respuestas a problemas operativos reales, como eficiencia de rutas, carga de trabajo de conductores o mantenimiento de vehículos. Para ello, se deben analizar los planes de ejecución, justificar los problemas de negocio que resuelve cada query y crear índices que mejoren el rendimiento.

A. Queries básicas:

1. Conocer la composición de la flota: A partir de esta consulta podemos ver que el tipo de vehículo que predomina en nuestra flota es el camión mediano, seguido por las motocicletas, las vans y los camiones grandes.

```sql
-- ====================================================
-- FLEETLOGIX - 12 QUERIES SQL
-- Organizadas por nivel de complejidad
-- ====================================================

-- ====================================================
-- QUERIES BÁSICAS (3 queries)
-- ====================================================

-- Query 1: Contar vehículos por tipo
-- Problema de negocio: Conocer la composición de la flota
SELECT
    vehicle_type,
    COUNT(*) as cantidad
FROM vehicles
GROUP BY vehicle_type
ORDER BY cantidad DESC;

-- Query 2: Listar conductores con licencia próxima a vencer
-- Problema de negocio: Prevenir problemas legales por licencias vencidas
SELECT
    first_name,
```

vehicles 1

SELECT vehicle_type, COUNT(*) as cantidad FROM v | Enter a SQL expression to filter results (use Ctrl+Space)

	vehicle_type	cantidad
1	Camión mediano	57
2	Camión grande	50
3	Van	48
4	Motocicleta	45

2. Listar conductores con licencia próxima a vencer.

```sql
-- Query 2: Listar conductores con licencia próxima a vencer
-- Problema de negocio: Prevenir problemas legales por licencias vencidas
SELECT
    first_name,
    last_name,
    license_number,
    license_expiry
FROM drivers
WHERE license_expiry < CURRENT_DATE + INTERVAL '30 days'
ORDER BY license_expiry;
```

drivers 1

SELECT first_name, last_name, license_number, licer | Enter a SQL expression to filter results (use Ctrl+Space)

	first_name	last_name	license_number	license_expiry
1	Vidal	Vall	LIC33817	2025-09-26
2	Ruperta	Zaragoza	LIC58572	2025-09-28
3	Olivia	Barrio	LIC28370	2025-10-01
4	Sandra	Asensio	LIC87534	2025-10-02
5	Leonel	Diaz	LIC62482	2025-10-03
6	Rufino	Priego	LIC93542	2025-10-04
7	Toni	Duran	LIC18733	2025-10-06
8	Pepita	Viñas	LIC57223	2025-10-06
9	Vinicio	Maestre	LIC95707	2025-10-06

A partir de esta consulta podemos prevenir problemas legales por licencias vencidas y poder iniciar los trámites de renovación de una forma anticipada. En este caso, se seleccionan los conductores cuya licencia esta próxima a vencer dentro de los próximos 30 dias.

3. Comprobar el número de viajes por status.
   La idea de esta query es monitoriear la situación de las operaciones en curso. En este caso llevamos 85 mil viajes completados, 5 mil cancelados y 10 mil en curso.

```sql
-- Query 3: Total de viajes por estado
-- Problema de negocio: Monitorear operaciones en curso
SELECT
    status,
    COUNT(*) as total_viajes
FROM trips
GROUP BY status;
```

trips 1

SELECT status, COUNT(*) as total_viajes FROM trips

Enter a SQL expression to filter results

	A-Z status	123 total_viajes
1	completed	84.855
2	cancelled	5.070
3	in_progress	10.075

B. Queries Intermedias:
   4. Identificar la demanda por ciudad. En este caso hay una gran demanda de envíos a Buenos Aires y Cordoba y en menor medida a Rosario y Mendoza

```sql
-- ================================================
-- QUERIES INTERMEDIAS (5 queries)
-- ================================================

-- Query 4: Total de entregas por ciudad destino en los últimos 2 meses
-- Problema de negocio: Identificar demanda por ciudad para planificación de recursos
SELECT
    r.destination_city,
    COUNT(DISTINCT t.trip_id) as total_viajes,
    COUNT(d.delivery_id) as total_entregas,
    SUM(d.package_weight_kg) as peso_total_kg
FROM routes r
INNER JOIN trips t ON r.route_id = t.route_id
INNER JOIN deliveries d ON t.trip_id = d.trip_id
WHERE t.departure_datetime >= CURRENT_DATE - INTERVAL '60 days'
GROUP BY r.destination_city
ORDER BY total_entregas DESC;
```

routes 1

SELECT r.destination_city, COUNT(DISTINCT t.trip_id

Enter a SQL expression to filter results (use Ctrl+Space)

	A-Z destination_city	123 total_viajes	123 total_entregas	123 peso_total_kg
1	Mar del Plata	1.147	4.635	5.069.260,12
2	Bahía Blanca	827	3.279	3.476.540,01
3	Buenos Aires	696	2.799	2.830.650,26
4	La Plata	684	2.752	2.785.818,67
5	Salta	671	2.683	2.633.244,63
6	Mendoza	646	2.559	2.791.324,96
7	Córdoba	592	2.369	2.338.945,21
8	Rosario	560	2.262	2.278.992,66

5. Conductores activos con más viajes completados. El objetivo es evaluar la eficiencia de los conductores en cuanto a cantidad de viajes realizados y completados.

```sql
-- Query 5: Conductores activos con cantidad de viajes completados
-- Problema de negocio: Evaluar carga de trabajo por conductor
SELECT
    d.driver_id,
    d.first_name || ' ' || d.last_name as nombre_completo,
    d.license_expiry,
    COUNT(t.trip_id) as viajes_totales,
    SUM(CASE WHEN t.status = 'completed' THEN 1 ELSE 0 END) as viajes_completados
FROM drivers d
LEFT JOIN trips t ON d.driver_id = t.driver_id
WHERE d.status = 'active'
GROUP BY d.driver_id, d.first_name, d.last_name, d.license_expiry
HAVING COUNT(t.trip_id) > 0
ORDER BY viajes_completados DESC;
```

drivers 1

Enter a SQL expression to filter results (use Ctrl+Space)

	driver_id	nombre_completo	license_expiry	viajes_totales	viajes_completados
1	179	Sofía Garriga	2025-11-19	600	521
2	118	Elisabet Alcolea	2026-07-12	594	519
3	317	Victorino Zurita	2026-09-16	602	515
4	175	Virginia Briones	2026-01-08	585	515
5	143	Mar Verdugo	2026-04-17	591	511
6	129	Hortensia Blanch	2026-09-02	587	509
7	313	Gloria Rosado	2026-07-11	599	509
8	166	Rufino Priego	2025-10-04	597	508
9	356	Felix Mayoral	2026-01-02	578	506
10	101	Emiliana Ortega	2026-09-17	589	505
11	365	Hernán Torrents	2026-09-12	583	505

6. Promedio de entregas por conductor en los últimos 6 meses.

Esta consulta nos permite evaluar la productividad por conductor en cuanto a entregas por viajes realizados. En este caso el conductor más eficiente es Sol Andreu con un promedio de 4,31 entregas por viaje realizado. Sin embargo, si evaluamos la eficiencia de entregas por día, Zoraida Soria lidera en la tabla, ya que tiene un promedio de 2.31 entregas por día.

```sql
-- Query 6: Promedio de entregas por conductor en los últimos 6 meses
-- Problema de negocio: Medir productividad individual de conductores
SELECT
    dr.driver_id,
    dr.first_name || ' ' || dr.last_name as conductor,
    COUNT(DISTINCT t.trip_id) as total_viajes,
    COUNT(d.delivery_id) as total_entregas,
    ROUND(COUNT(d.delivery_id)::NUMERIC / NULLIF(COUNT(DISTINCT t.trip_id), 0), 2) as promedio_entregas_por_viaje,
    ROUND(COUNT(d.delivery_id)::NUMERIC / 180, 2) as promedio_entregas_diarias
FROM drivers dr
INNER JOIN trips t ON dr.driver_id = t.driver_id
INNER JOIN deliveries d ON t.trip_id = d.trip_id
WHERE t.departure_datetime >= CURRENT_DATE - INTERVAL '6 months'
    AND t.status = 'completed'
GROUP BY dr.driver_id, dr.first_name, dr.last_name
HAVING COUNT(DISTINCT t.trip_id) >= 10
ORDER BY promedio_entregas_por_viaje DESC;
```

drivers 1

Enter a SQL expression to filter results (use Ctrl+Space)

	driver_id	conductor	total_viajes	total_entregas	promedio_entregas_por_viaje	promedio_entregas_diarias
1	272	Sol Andreu	68	295	4,34	1,6
2	193	Javiera Ortuño	93	402	4,32	2,2
3	206	Zoraida Soria	97	416	4,29	2,3
4	1	Nidia Quintana	82	347	4,23	1,9
5	5	Cruz Almansa	73	308	4,22	1,7
6	259	Horacio Carvajal	94	396	4,21	2,
7	83	Sonia Peinado	90	378	4,2	2,
8	128	Serafina Madrigal	75	315	4,2	1,7
9	224	Penita Viñas	95	398	4,19	2,2

C. Queries Complejas:

9. Costo de mantenimiento por kilómetro recorrido. Permite evaluar el costo – beneficio de cada tipo de vehículo.

```sql
-- Query 9: Costo de mantenimiento por kilómetro recorrido
-- Problema de negocio: Evaluar costo-beneficio de cada tipo de vehículo
WITH vehicle_metrics AS (
    SELECT
        v.vehicle_id,
        v.vehicle_type,
        v.license_plate,
        COUNT(DISTINCT t.trip_id) as total_viajes,
        SUM(r.distance_km) as km_totales,
        SUM(m.cost) as costo_mantenimiento_total,
        COUNT(DISTINCT m.maintenance_id) as cantidad_mantenimientos
    FROM vehicles v
    LEFT JOIN trips t ON v.vehicle_id = t.vehicle_id
    LEFT JOIN routes r ON t.route_id = r.route_id
    LEFT JOIN maintenance m ON v.vehicle_id = m.vehicle_id
    WHERE t.status = 'completed'
    GROUP BY v.vehicle_id, v.vehicle_type, v.license_plate
)
SELECT
    vehicle_type,
    COUNT(vehicle_id) as cantidad_vehiculos,
    SUM(total_viajes) as viajes_totales,
    SUM(km_totales) as kilometros_totales,
    SUM(costo_mantenimiento_total) as costo_total_mantenimiento,
```

	vehicle_type	cantidad_vehiculos	viajes_totales	kilometros_totales	costo_total_mantenimiento	costo_por_km
1	Motocicleta	17	22.994	429.150.641,48	12.956.015.508,78	30,19
2	Van	9	11.930	216.185.583,45	6.336.351.248,39	29,31
3	Camión mediano	21	28.315	459.396.714,65	13.168.018.457,8	28,66
4	Camión grande	16	21.616	345.214.465,06	9.853.199.197,96	28,54

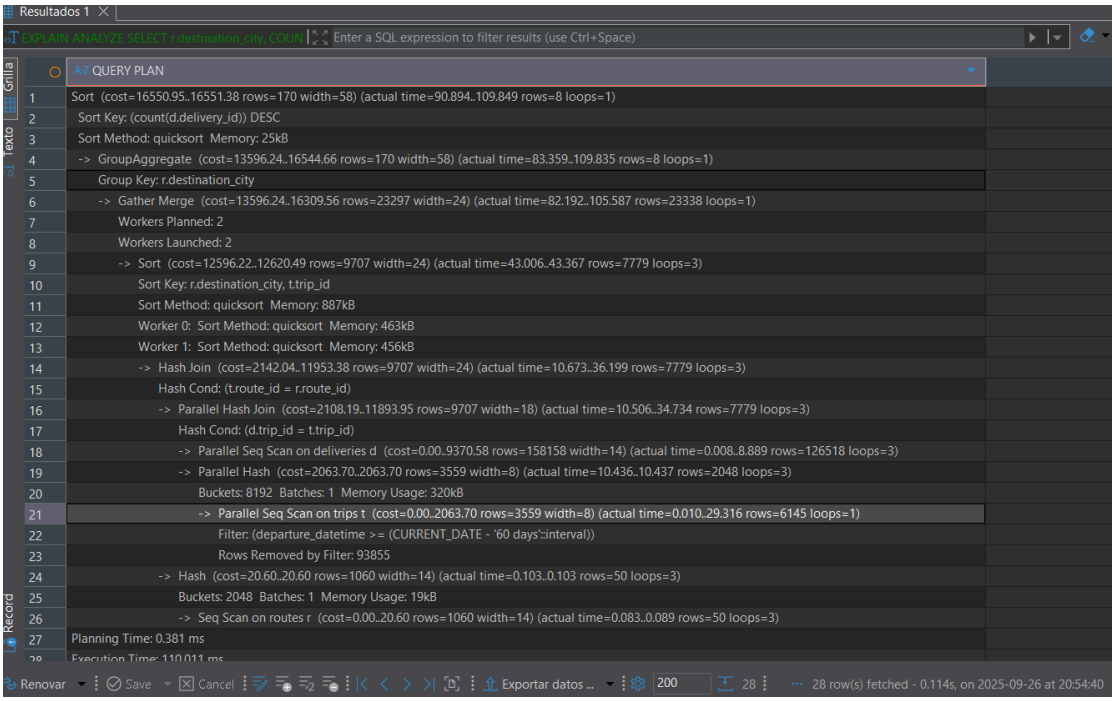
12. Entregas por día y horario de semana. Permite optimizar horarios de operación personal.

```sql
-- Query 12: Pivot de entregas por hora y día de la semana
-- Problema de negocio: Optimizar horarios de operación y personal
WITH entregas_por_hora_dia AS (
    SELECT
        EXTRACT(DOW FROM scheduled_datetime) as dia_semana,
        EXTRACT(HOUR FROM scheduled_datetime) as hora,
        COUNT(*) as cantidad_entregas
    FROM deliveries
    WHERE scheduled_datetime >= CURRENT_DATE - INTERVAL '60 days'
    GROUP BY EXTRACT(DOW FROM scheduled_datetime), EXTRACT(HOUR FROM scheduled_datetime)
)
SELECT
    hora,
    SUM(CASE WHEN dia_semana = 0 THEN cantidad_entregas ELSE 0 END) as domingo,
    SUM(CASE WHEN dia_semana = 1 THEN cantidad_entregas ELSE 0 END) as lunes,
    SUM(CASE WHEN dia_semana = 2 THEN cantidad_entregas ELSE 0 END) as martes,
    SUM(CASE WHEN dia_semana = 3 THEN cantidad_entregas ELSE 0 END) as miercoles,
    SUM(CASE WHEN dia_semana = 4 THEN cantidad_entregas ELSE 0 END) as jueves,
    SUM(CASE WHEN dia_semana = 5 THEN cantidad_entregas ELSE 0 END) as viernes,
    SUM(CASE WHEN dia_semana = 6 THEN cantidad_entregas ELSE 0 END) as sabado,
    SUM(cantidad_entregas) as total_semana
FROM entregas_por_hora_dia
WHERE hora BETWEEN 6 AND 22
GROUP BY hora
ORDER BY hora;
```

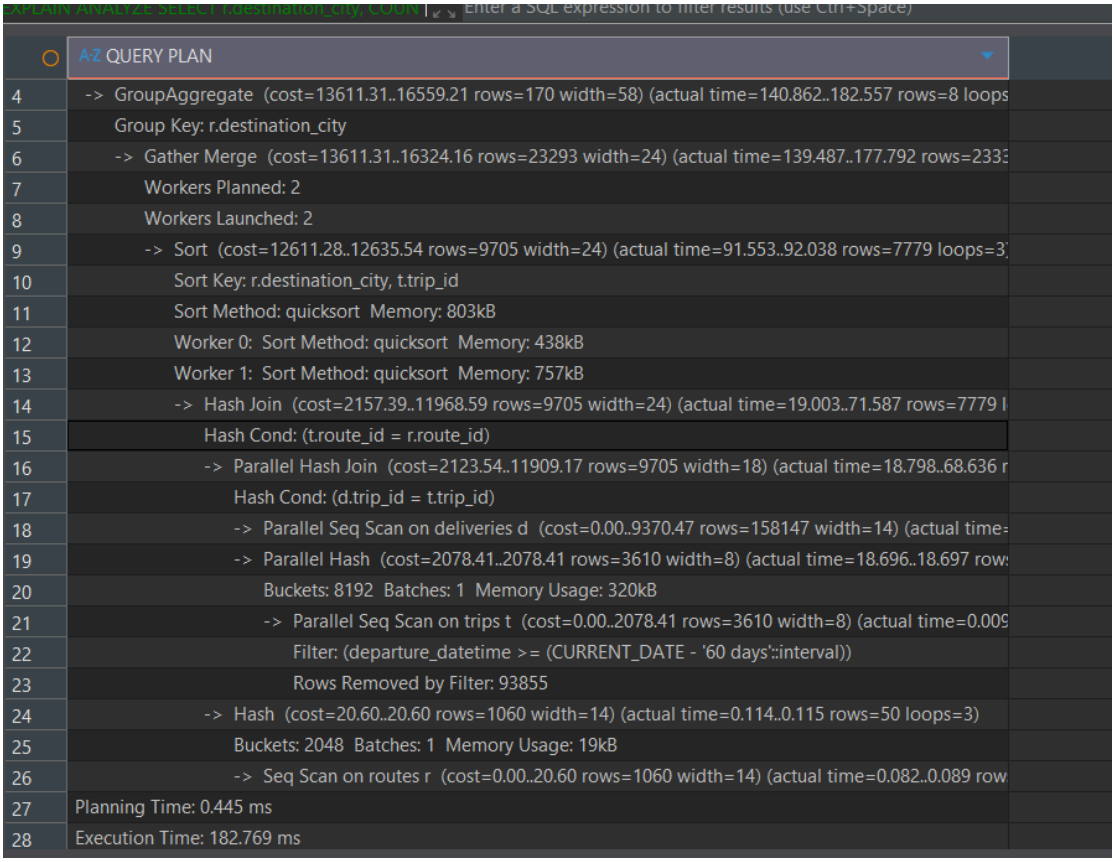
	hora	domingo	lunes	martes	miercoles	jueves	viernes	sabado	total_semana
1	6	102	81	112	90	93	100	92	670
2	7	86	103	111	117	89	102	101	709
3	8	104	111	139	113	110	127	105	809
4	9	121	137	168	140	128	141	132	967
5	10	141	198	186	176	163	154	148	1.166
6	11	157	190	181	187	163	173	148	1.199
7	12	164	207	180	161	179	132	138	1.161

Query 4

- Sin índice 1



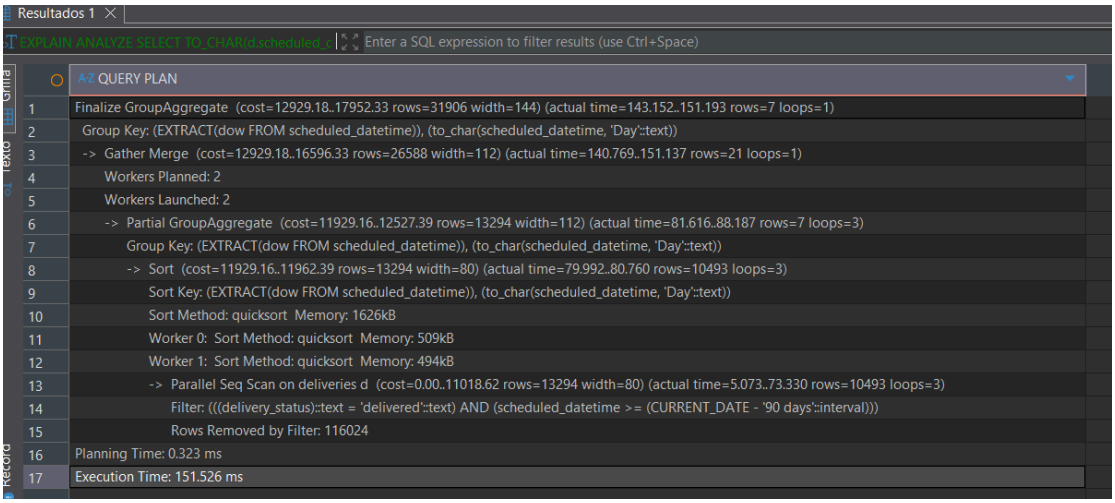
- Con índice 1



	QUERY PLAN
4	-> GroupAggregate (cost=13611.31..16559.21 rows=170 width=58) (actual time=140.862..182.557 rows=8 loops
5	Group Key: r.destination_city
6	-> Gather Merge (cost=13611.31..16324.16 rows=23293 width=24) (actual time=139.487..177.792 rows=233
7	Workers Planned: 2
8	Workers Launched: 2
9	-> Sort (cost=12611.28..12635.54 rows=9705 width=24) (actual time=91.553..92.038 rows=7779 loops=3
10	Sort Key: r.destination_city, t.trip_id
11	Sort Method: quicksort Memory: 803kB
12	Worker 0: Sort Method: quicksort Memory: 438kB
13	Worker 1: Sort Method: quicksort Memory: 757kB
14	-> Hash Join (cost=2157.39..11968.59 rows=9705 width=24) (actual time=19.003..71.587 rows=7779 l
15	Hash Cond: (t.route_id = r.route_id)
16	-> Parallel Hash Join (cost=2123.54..11909.17 rows=9705 width=18) (actual time=18.798..68.636 r
17	Hash Cond: (d.trip_id = t.trip_id)
18	-> Parallel Seq Scan on deliveries d (cost=0.00..9370.47 rows=158147 width=14) (actual time=
19	-> Parallel Hash (cost=2078.41..2078.41 rows=3610 width=8) (actual time=18.696..18.697 rows
20	Buckets: 8192 Batches: 1 Memory Usage: 320kB
21	-> Parallel Seq Scan on trips t (cost=0.00..2078.41 rows=3610 width=8) (actual time=0.009
22	Filter: (departure_datetime >= (CURRENT_DATE - '60 days'::interval))
23	Rows Removed by Filter: 93855
24	-> Hash (cost=20.60..20.60 rows=1060 width=14) (actual time=0.114..0.115 rows=50 loops=3)
25	Buckets: 2048 Batches: 1 Memory Usage: 19kB
26	-> Seq Scan on routes r (cost=0.00..20.60 rows=1060 width=14) (actual time=0.082..0.089 row
27	Planning Time: 0.445 ms
28	Execution Time: 182.769 ms

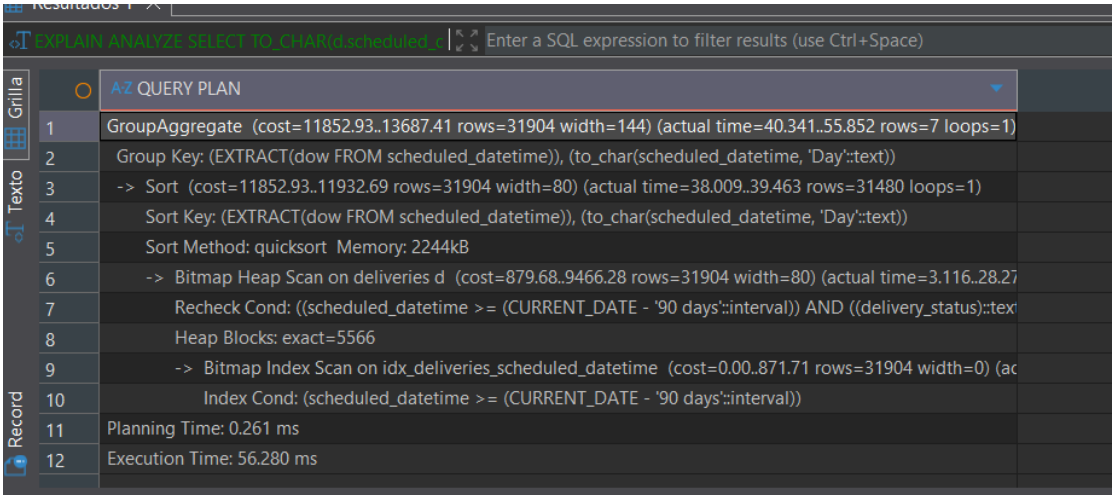
Query 8 –

- Sin indice 2



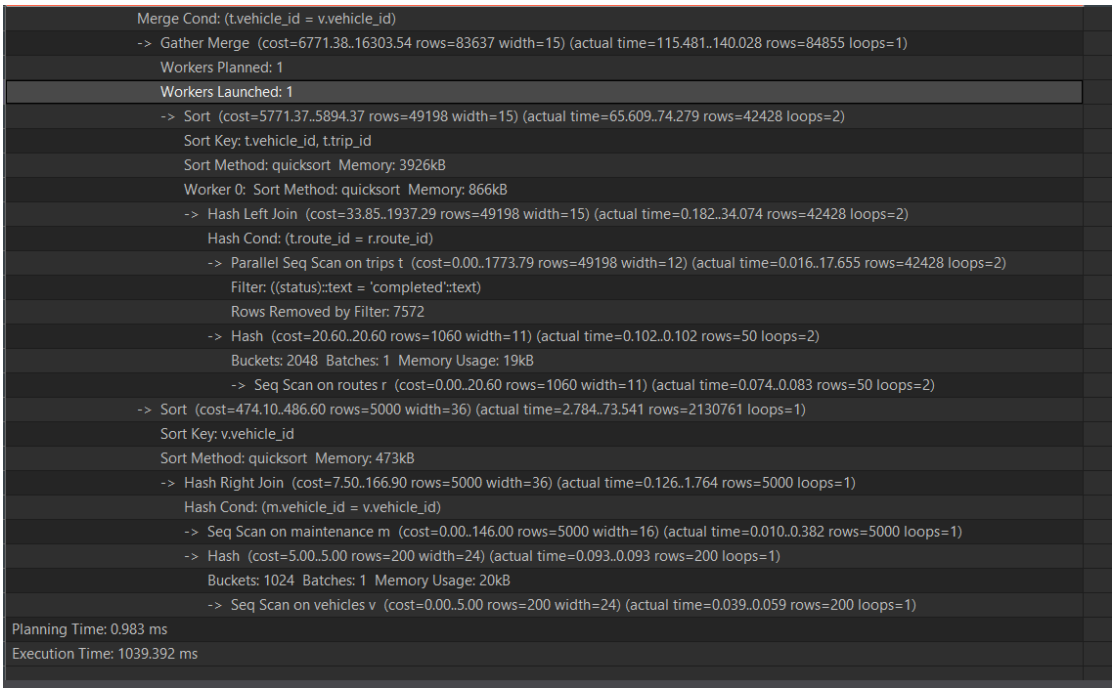
	QUERY PLAN
1	Finalize GroupAggregate (cost=12929.18..17952.33 rows=31906 width=144) (actual time=143.152..151.193 rows=7 loops=1)
2	Group Key: (EXTRACT(dow FROM scheduled_datetime)), (to_char(scheduled_datetime, 'Day'::text))
3	-> Gather Merge (cost=12929.18..16596.33 rows=26588 width=112) (actual time=140.769..151.137 rows=21 loops=1)
4	Workers Planned: 2
5	Workers Launched: 2
6	-> Partial GroupAggregate (cost=11929.16..12527.39 rows=13294 width=112) (actual time=81.616..88.187 rows=7 loops=3)
7	Group Key: (EXTRACT(dow FROM scheduled_datetime)), (to_char(scheduled_datetime, 'Day'::text))
8	-> Sort (cost=11929.16..11962.39 rows=13294 width=80) (actual time=79.992..80.760 rows=10493 loops=3)
9	Sort Key: (EXTRACT(dow FROM scheduled_datetime)), (to_char(scheduled_datetime, 'Day'::text))
10	Sort Method: quicksort Memory: 1626kB
11	Worker 0: Sort Method: quicksort Memory: 509kB
12	Worker 1: Sort Method: quicksort Memory: 494kB
13	-> Parallel Seq Scan on deliveries d (cost=0.00..11018.62 rows=13294 width=80) (actual time=5.073..73.330 rows=10493 loops=3)
14	Filter: (((delivery_status)::text = 'delivered'::text) AND (scheduled_datetime >= (CURRENT_DATE - '90 days'::interval)))
15	Rows Removed by Filter: 116024
16	Planning Time: 0.323 ms
17	Execution Time: 151.526 ms

- Con indice 2



Query 9

- Sin indice 3



- Con Indice 3

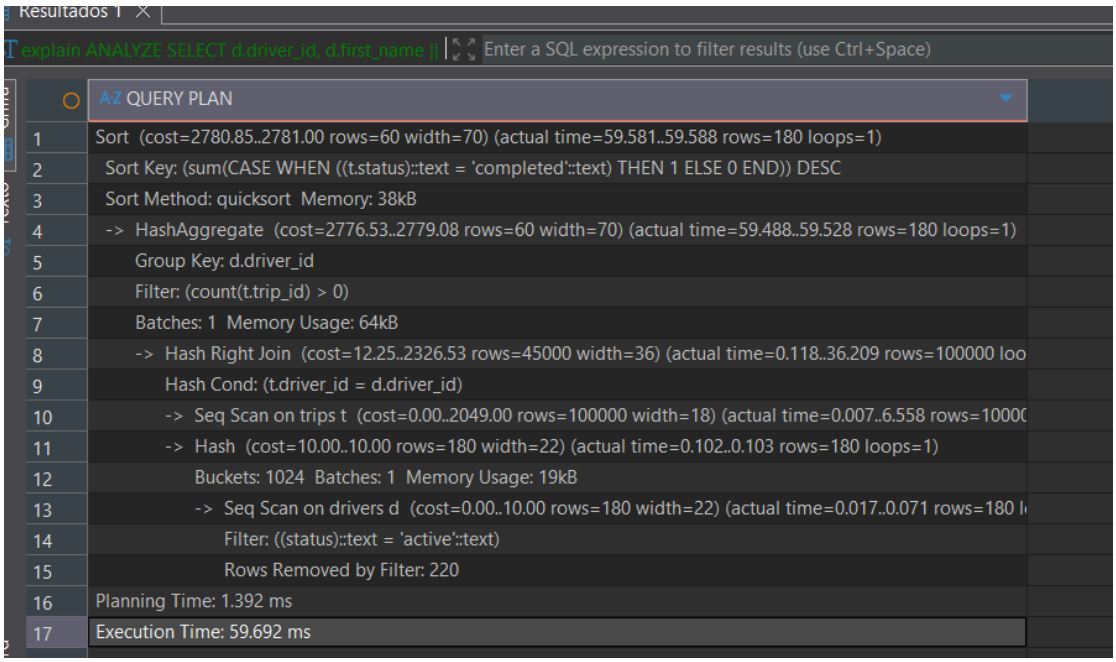
```
                    Workers Planned: 1
                    Workers Launched: 1
                    ->  Sort  (cost=5985.89..6110.67 rows=49912 width=35) (actual time=68.856..75.883 rows=4
                        Sort Key: v.vehicle_id, t.trip_id
                        Sort Method: external merge  Disk: 2480kB
                        Worker 0:  Sort Method: quicksort  Memory: 3145kB
                        ->  Hash Left Join  (cost=41.35..2090.99 rows=49912 width=35) (actual time=0.225..28.4
                            Hash Cond: (t.route_id = r.route_id)
                            ->  Hash Join  (cost=7.50..1925.59 rows=49912 width=32) (actual time=0.157..19.769
                                Hash Cond: (t.vehicle_id = v.vehicle_id)
                                ->  Parallel Seq Scan on trips t  (cost=0.00..1784.29 rows=49912 width=12) (actu
                                    Filter: ((status)::text = 'completed'::text)
                                    Rows Removed by Filter: 7572
                                ->  Hash  (cost=5.00..5.00 rows=200 width=24) (actual time=0.098..0.099 rows=2
                                    Buckets: 1024  Batches: 1  Memory Usage: 20kB
                                    ->  Seq Scan on vehicles v  (cost=0.00..5.00 rows=200 width=24) (actual time
                            ->  Hash  (cost=20.60..20.60 rows=1060 width=11) (actual time=0.060..0.061 rows=5
                                Buckets: 2048  Batches: 1  Memory Usage: 19kB
                                ->  Seq Scan on routes r  (cost=0.00..20.60 rows=1060 width=11) (actual time=0.
                ->  Memoize  (cost=0.29..3.09 rows=25 width=16) (actual time=0.000..0.001 rows=25 loops=84
                    Cache Key: v.vehicle_id
                    Cache Mode: logical
                    Hits: 84792  Misses: 63  Evictions: 0  Overflows: 0  Memory Usage: 83kB
                    ->  Index Scan using idx_maintenance_vehicle_cost on maintenance m  (cost=0.28..3.08 row
                        Index Cond: (vehicle_id = v.vehicle_id)
Planning Time: 0.683 ms
Execution Time: 899.377 ms
```

Query 5

- sin indice 4

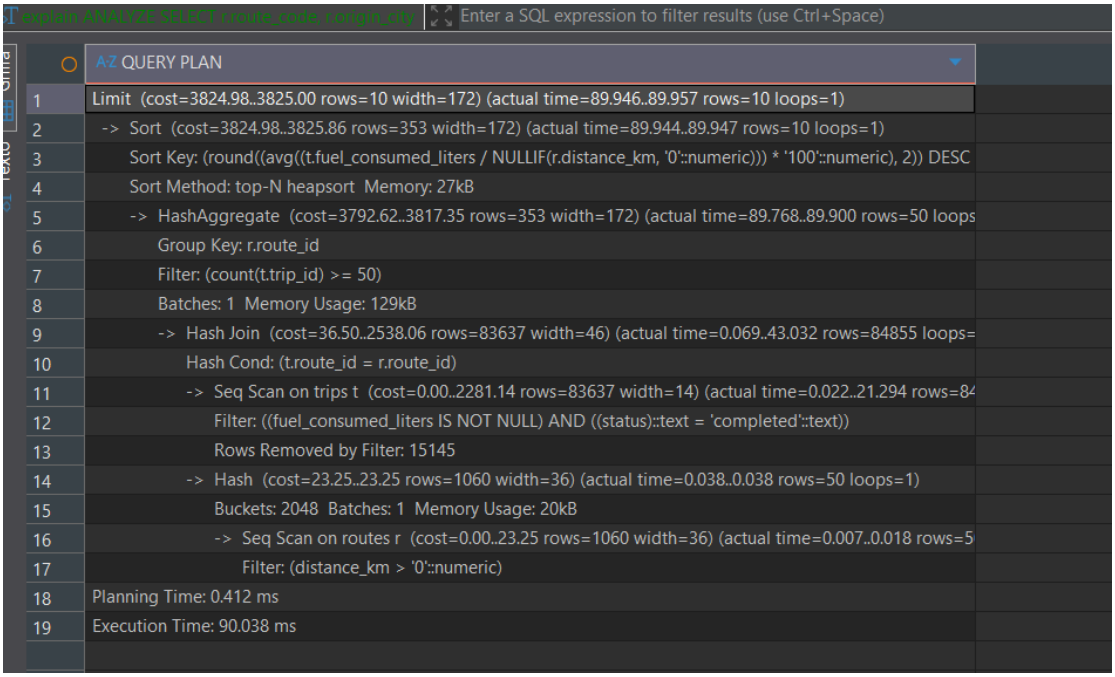
```
explain ANALYZE SELECT d.driver_id, d.first_name ||  Enter a SQL expression to filter results (use Ctrl+Space)

   A-Z QUERY PLAN
1  Sort  (cost=2756.34..2756.49 rows=60 width=70) (actual time=61.973..61.980 rows=180 loops=1)
2    Sort Key: (sum(CASE WHEN ((t.status)::text = 'completed'::text) THEN 1 ELSE 0 END)) DESC
3    Sort Method: quicksort  Memory: 38kB
4    ->  HashAggregate  (cost=2752.02..2754.57 rows=60 width=70) (actual time=61.889..61.926 rows=180 loops=1)
5      Group Key: d.driver_id
6      Filter: (count(t.trip_id) > 0)
7      Batches: 1  Memory Usage: 64kB
8      ->  Hash Right Join  (cost=12.25..2308.45 rows=44357 width=36) (actual time=0.100..37.230 rows=100000 loo
9        Hash Cond: (t.driver_id = d.driver_id)
10       ->  Seq Scan on trips t  (cost=0.00..2034.71 rows=98571 width=18) (actual time=0.005..7.022 rows=100000
11       ->  Hash  (cost=10.00..10.00 rows=180 width=22) (actual time=0.088..0.089 rows=180 loops=1)
12         Buckets: 1024  Batches: 1  Memory Usage: 19kB
13         ->  Seq Scan on drivers d  (cost=0.00..10.00 rows=180 width=22) (actual time=0.013..0.064 rows=180 l
14           Filter: ((status)::text = 'active'::text)
15           Rows Removed by Filter: 220
16 Planning Time: 0.236 ms
17 Execution Time: 62.043 ms
```

- Con indice 4

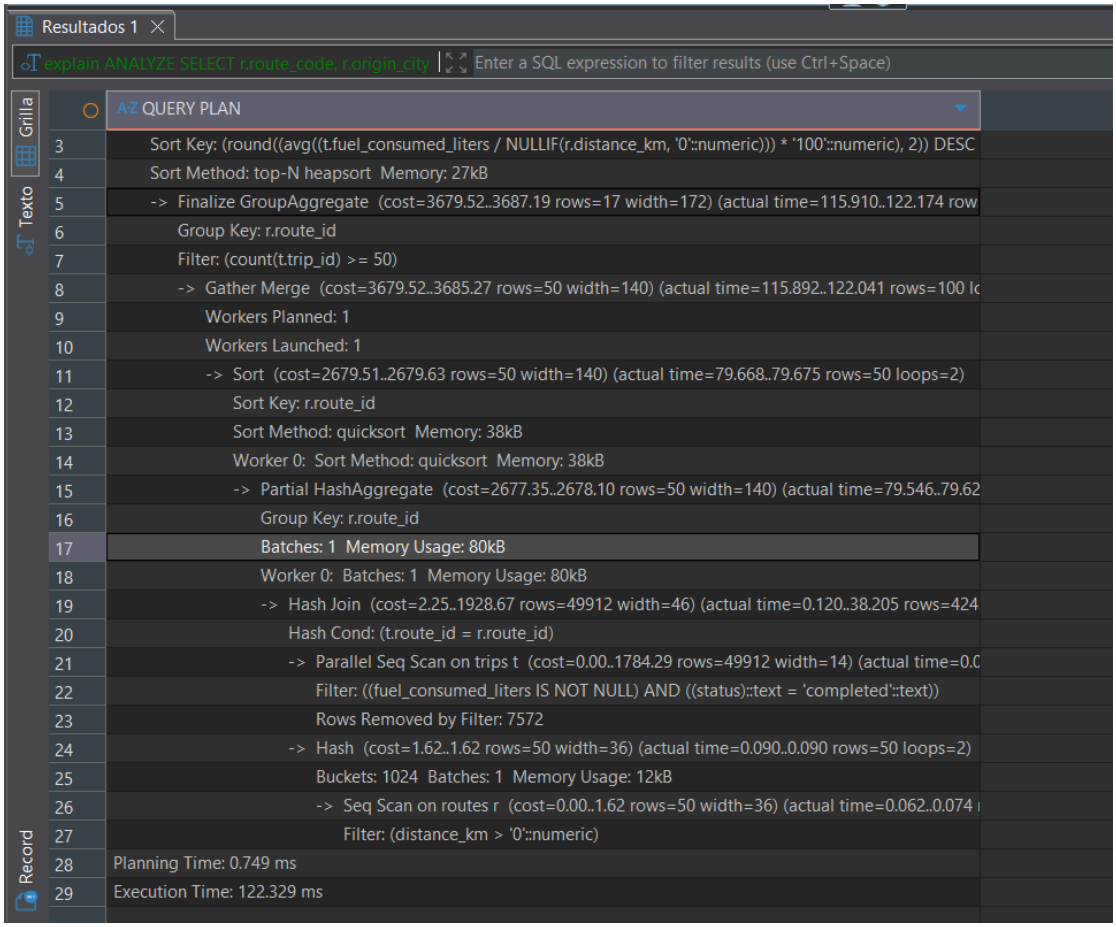


Query 7

- sin indice 5



- Con Indice 5—

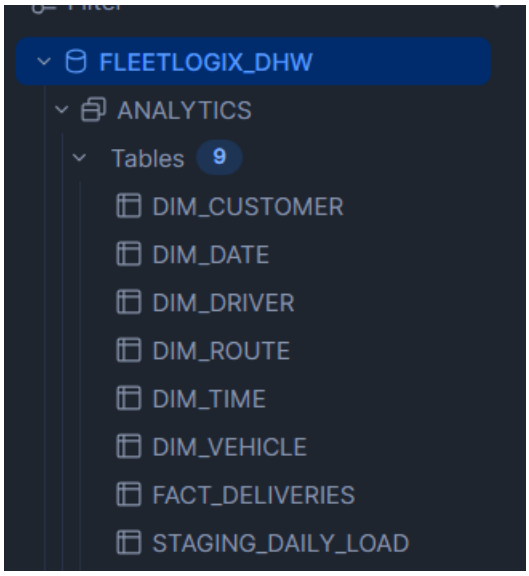


Aclaración: en algunos casos no se nota una diferencia significativa en cuanto al tiempo y/o proceso de ejecución porque la base es relativamente pequeña en términos de información almacenada.

Avance 3: Análisis de Snowflake

Para poder armar el flujo de datos de nuestra base de Postgre a Snowflake lo primero que hay que hacer es crear la estructura de nuestro sistema OLTP:

```
--- CREAMOS UN VIRTUAL WAREHOUSE ---
USE ROLE ACCOUNTADMIN;
CREATE WAREHOUSE IF NOT EXISTS FLEETLOGIX_WH WITH WAREHOUSE_SIZE = 'XSMALL' AUTO_SUSPEND = 60;

---CREAMOS LA BASE DE FLEETLOGIX---
CREATE DATABASE FleetLogix_dhw;
USE DATABASE FleetLogix_dhw;

---CREAMOS EL ESQUEMA DE LA BASE DE FLEETLOGIX---
CREATE SCHEMA IF NOT EXISTS ANALYTICS;
USE SCHEMA ANALYTICS;

```



Luego creamos las conexiones entre las herramientas

```
import pandas as pd
import numpy as np
from datetime import datetime
import psycopg2
import snowflake.connector

# PostgreSQL
conn_pg = psycopg2.connect(
    host="localhost",
    database="fleetlogix",
    user="postgres",
    password="Spriest123"
)
cur_pg = conn_pg.cursor()

# Snowflake
conn_sf = snowflake.connector.connect(
    user='ENZOZAMBON',
    password='AdiiraelSpriest12345',
    account='GTUWIRG-PU45327',
    warehouse='FLEETLOGIX_WH',
    database='FleetLogix_dhw',
    schema='ANALYTICS'
)
cur_sf = conn_sf.cursor()
```

[71]

```
INFO:snowflake.connector.connection:Snowflake Connector for Python Version: 4.0.0, Python Version: 3.13.0, Platform: Windows-10-10.0.19043-SP0
INFO:snowflake.connector.connection:Connecting to GLOBAL Snowflake domain
```

```
def insert_in_batches(cursor, sql, data, batch_size=5000, table_name=""):
    for i in range(0, len(data), batch_size):
        batch = data[i:i+batch_size]
        cursor.executemany(sql, batch)
        conn_sf.commit()
        print(f"[{datetime.now()}] {table_name} - batch {i//batch_size+1} success - {len(batch)} registros")
```

[72]

Posteriormente cargamos la información de las tablas en multiples dataframes:

```
df_deliveries = pd.read_sql("SELECT * FROM deliveries", conn_pg)
df_trip = pd.read_sql("SELECT * FROM trips", conn_pg)
df_vehicle = pd.read_sql("SELECT * FROM vehicles", conn_pg)
df_driver = pd.read_sql("SELECT * FROM drivers", conn_pg)
df_route = pd.read_sql("SELECT * FROM routes", conn_pg)
df_maint = pd.read_sql("SELECT vehicle_id, MAX(maintenance_date) AS last_maintenance_date FROM maintenance GROUP BY vehicle_id", conn_pg)
```

[73]

```
C:\Users\Enzo\AppData\Local\Temp\ipykernel_19296\2084301068.py:1: UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or d
  df_deliveries = pd.read_sql("SELECT * FROM deliveries", conn_pg)
C:\Users\Enzo\AppData\Local\Temp\ipykernel_19296\2084301068.py:2: UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or d
  df_trip = pd.read_sql("SELECT * FROM trips", conn_pg)
C:\Users\Enzo\AppData\Local\Temp\ipykernel_19296\2084301068.py:3: UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or d
  df_vehicle = pd.read_sql("SELECT * FROM vehicles", conn_pg)
C:\Users\Enzo\AppData\Local\Temp\ipykernel_19296\2084301068.py:4: UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or d
  df_driver = pd.read_sql("SELECT * FROM drivers", conn_pg)
C:\Users\Enzo\AppData\Local\Temp\ipykernel_19296\2084301068.py:5: UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or d
  df_route = pd.read_sql("SELECT * FROM routes", conn_pg)
C:\Users\Enzo\AppData\Local\Temp\ipykernel_19296\2084301068.py:6: UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or d
  df_maint = pd.read_sql("SELECT vehicle_id, MAX(maintenance_date) AS last_maintenance_date FROM maintenance GROUP BY vehicle_id", conn_pg)
```

```
df_deliveries['scheduled_datetime'] = pd.to_datetime(df_deliveries['scheduled_datetime'])
df_trip['departure_datetime'] = pd.to_datetime(df_trip['departure_datetime'])
df_trip['arrival_datetime'] = pd.to_datetime(df_trip['arrival_datetime'])
```

[ ]

Y poblamos las tablas de Snowflake, procesamiento de datos mediante acorde a la estructura de las tablas previamente creadas

```python
# ---------------------------- Dimensión VEHICLES ----------------------------
# Convertir acquisition_date a datetime
df_vehicle['acquisition_date'] = pd.to_datetime(df_vehicle['acquisition_date'], errors='coerce')

# Traer la última fecha de mantenimiento por vehículo
df_maint['last_maintenance_date'] = pd.to_datetime(df_maint['last_maintenance_date'], errors='coerce')
df_maint_last = df_maint.groupby('vehicle_id', as_index=False)['last_maintenance_date'].max()

# Merge seguro con df_vehicle
if 'last_maintenance_date' in df_vehicle.columns:
    df_vehicle = df_vehicle.drop(columns=['last_maintenance_date'])

df_vehicle = df_vehicle.merge(df_maint_last, on='vehicle_id', how='left')

# Calcular edad en meses
df_vehicle['age_months'] = ((pd.Timestamp.today() - df_vehicle['acquisition_date']).dt.days // 30).fillna(0).astype(int)

# Asegurarse de que last_maintenance_date exista
if 'last_maintenance_date' not in df_vehicle.columns:
    df_vehicle['last_maintenance_date'] = pd.NaT

# Convertir fechas a string para Snowflake
df_vehicle['acquisition_date'] = df_vehicle['acquisition_date'].dt.strftime('%Y-%m-%d %H:%M:%S')
df_vehicle['last_maintenance_date'] = df_vehicle['last_maintenance_date'].dt.strftime('%Y-%m-%d %H:%M:%S')

# Preparar registros para insert
vehicle_records = list(df_vehicle[['vehicle_id','license_plate','vehicle_type','capacity_kg','fuel_type',
                                   'acquisition_date','age_months','status','last_maintenance_date']].itertuples(index=False, name=None))

# Insertar en batches
insert_in_batches(cur_sf, """
INSERT INTO dim_vehicle (vehicle_id, license_plate, vehicle_type, capacity_kg, fuel_type, acquisition_date, age_months, status, last_maintenance_date)
VALUES (%s,%s,%s,%s,%s,%s,%s,%s,%s)
""", vehicle_records, batch_size=5000, table_name="dim_vehicle")
```

[75]

```
[2025-10-13 22:38:50.947028] dim_vehicle - batch 1 success - 200 registros
```

```python
# ---------------------------- Dimensión DRIVERS ----------------------------

# Convertir fechas a datetime
df_driver['hire_date'] = pd.to_datetime(df_driver['hire_date'], errors='coerce')
df_driver['license_expiry'] = pd.to_datetime(df_driver['license_expiry'], errors='coerce')

# Calcular experiencia en meses
df_driver['experience_months'] = ((pd.Timestamp.today() - df_driver['hire_date']).dt.days // 30).fillna(0).astype(int)

# Crear full_name
df_driver['full_name'] = df_driver['first_name'].fillna('') + ' ' + df_driver['last_name'].fillna('')

# Contar viajes por conductor
df_trip_counts = df_trip.groupby('driver_id').size().reset_index(name='total_trips')

# Hacer merge
if 'total_trips' in df_driver.columns:
    df_driver = df_driver.drop(columns=['total_trips'])

df_driver = df_driver.merge(df_trip_counts, on='driver_id', how='left')

# Rellenar NaN con 0
df_driver['total_trips'] = df_driver['total_trips'].fillna(0)
df_driver['completed_trips'] = df_driver['total_trips']

# Categoría de desempeño evitando división por 0
df_driver['performance_category'] = np.where(
    df_driver['total_trips'] == 0, 'Bajo',  # Si no hizo viajes
    np.where(df_driver['completed_trips']/df_driver['total_trips'] > 0.7, 'Alto',
             np.where(df_driver['completed_trips']/df_driver['total_trips'] > 0.5, 'Medio','Bajo'))
)

# Convertir fechas a string para Snowflake (maneja NaT)
df_driver['hire_date'] = df_driver['hire_date'].dt.strftime('%Y-%m-%d %H:%M:%S')
df_driver['license_expiry'] = df_driver['license_expiry'].dt.strftime('%Y-%m-%d %H:%M:%S')

# Reemplazar NaN por None para Snowflake
df_driver = df_driver.where(pd.notnull(df_driver), None)

# Preparar registros
driver_records = list(df_driver[['driver_id', 'employee_code','full_name','license_number','license_expiry',
                                 'phone','hire_date','experience_months','status','performance_category']].itertuples(index=False, name=None))

# Insertar en batches
insert_in_batches(cur_sf, """
INSERT INTO dim_driver (driver_id, employee_code, full_name, license_number, license_expiry, phone, hire_date, experience_months, status, performance_category)
VALUES (%s,%s,%s,%s,%s,%s,%s,%s,%s,%s)

""", driver_records, batch_size=5000, table_name="dim_driver")
```

[76]

```
# ------------------------- Dimensión ROUTES -------------------------
# Llenar valores nulos en numéricos
df_route[['distance_km','estimated_duration_hours','toll_cost']] = df_route[['distance_km','estimated_duration_hours','toll_cost']].fillna(0)

# Asegurar tipos
df_route['distance_km'] = df_route['distance_km'].astype(float)
df_route['estimated_duration_hours'] = df_route['estimated_duration_hours'].astype(float)
df_route['toll_cost'] = df_route['toll_cost'].astype(float)

# Renombrar columnas si es necesario
df_route = df_route.rename(columns={
    'origin': 'origin_city',
    'destination': 'destination_city'
})

# Preparar registros
route_records = list(df_route[['route_id', 'route_code','origin_city','destination_city','distance_km','estimated_duration_hours','toll_cost']].itertuples(index=False, name=None))

# Insertar en batches
insert_in_batches(cur_sf, """
INSERT INTO dim_route (route_id, route_code, origin_city, destination_city, distance_km, estimated_duration_hours, toll_cost)
VALUES (%s,%s,%s,%s,%s,%s,%s)
""", route_records, batch_size=5000, table_name="dim_route")
```

```
[77]
...  [2025-10-13 22:38:55.344954] dim_route - batch 1 success - 50 registros
```

```
# ------------------------- Dimensión CUSTOMER -------------------------
df_trip = df_trip.merge(df_route[['route_id','destination_city']], on='route_id', how='left')
df_customer = df_deliveries.merge(df_trip[['trip_id','destination_city']], on='trip_id', how='left')
df_customer_group = df_customer.groupby(['customer_name','destination_city']).size().reset_index(name='total_deliveries')

def cat_func(x):
    if x==1: return 'Ocasional'
    elif x==2: return 'Regular'
    else: return 'Habitual'

df_customer_group['customer_category'] = df_customer_group['total_deliveries'].apply(cat_func)
customers_dim = [(i+1, row['customer_name'], row['destination_city'], datetime.today(), row['total_deliveries'], row['customer_category'])
                 for i,row in df_customer_group.iterrows()]

insert_in_batches(cur_sf, """
INSERT INTO dim_customer (customer_key, customer_name, city, first_delivery_date, total_deliveries, customer_category)
VALUES (%s,%s,%s,%s,%s,%s)
""", customers_dim, batch_size=5000, table_name="dim_customer")
```

```
[ ]
...  [2025-10-13 22:39:11.051824] dim_customer - batch 1 success - 5000 registros
[2025-10-13 22:39:13.785869] dim_customer - batch 2 success - 5000 registros
[2025-10-13 22:39:16.333784] dim_customer - batch 3 success - 5000 registros
[2025-10-13 22:39:19.274304] dim_customer - batch 4 success - 5000 registros
[2025-10-13 22:39:22.730543] dim_customer - batch 5 success - 5000 registros
[2025-10-13 22:39:26.134172] dim_customer - batch 6 success - 5000 registros
[2025-10-13 22:39:30.029869] dim_customer - batch 7 success - 5000 registros
[2025-10-13 22:39:33.100676] dim_customer - batch 8 success - 5000 registros
[2025-10-13 22:39:35.737353] dim_customer - batch 9 success - 5000 registros
[2025-10-13 22:39:38.592063] dim_customer - batch 10 success - 5000 registros
[2025-10-13 22:39:41.495406] dim_customer - batch 11 success - 5000 registros
[2025-10-13 22:39:44.085903] dim_customer - batch 12 success - 5000 registros
[2025-10-13 22:39:46.971557] dim_customer - batch 13 success - 5000 registros
[2025-10-13 22:39:49.855782] dim_customer - batch 14 success - 5000 registros
[2025-10-13 22:39:52.794118] dim_customer - batch 15 success - 5000 registros
[2025-10-13 22:39:55.579416] dim_customer - batch 16 success - 5000 registros
[2025-10-13 22:39:58.100893] dim_customer - batch 17 success - 5000 registros
[2025-10-13 22:40:01.778784] dim_customer - batch 18 success - 5000 registros
[2025-10-13 22:40:04.689977] dim_customer - batch 19 success - 5000 registros
[2025-10-13 22:40:07.244356] dim_customer - batch 20 success - 5000 registros
[2025-10-13 22:40:10.068254] dim_customer - batch 21 success - 5000 registros
[2025-10-13 22:40:12.481030] dim_customer - batch 22 success - 5000 registros
[2025-10-13 22:40:15.561460] dim_customer - batch 23 success - 5000 registros
[2025-10-13 22:40:18.576553] dim_customer - batch 24 success - 5000 registros
[2025-10-13 22:40:21.834860] dim_customer - batch 25 success - 5000 registros
[2025-10-13 22:40:22.723200] dim_customer - batch 26 success - 828 registros
```

```python
# ------------------------- Dimensión DATE -------------------------
all_dates = set(df_deliveries['scheduled_datetime'].dt.date)
dim_date = []

for dt in all_dates:
    dim_date.append((
        int(dt.strftime("%Y%m%d")),
        datetime.combine(dt, datetime.min.time()),
        dt.isoweekday(),
        dt.strftime("%A"),
        dt.day,
        dt.timetuple().tm_yday,
        dt.isocalendar()[1],
        dt.month,
        dt.strftime("%B"),
        (dt.month-1)//3+1,
        dt.year,
        dt.weekday()>=5,
        0, None,
        (dt.month-1)//3+1,
        dt.year
    ))

insert_in_batches(cur_sf, """
INSERT INTO dim_date (date_key, full_date, day_of_week, day_name, day_of_month, day_of_year, week_of_year, month_num, month_name, quarter, year, is_weekend, is_holiday, holiday_name, fiscal_quarter, fiscal_year)
VALUES (%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s)
""", dim_date, batch_size=5000, table_name="dim_date")
```

```
[79]
[2025-10-13 22:40:24.145377] dim_date - batch 1 success - 1018 registros
```

```python
# ------------------------- Dimensión TIME -------------------------
dim_time = []

for h in range(24):
    for m in range(60):
        time_key = h*10000 + m*100
        am_pm = 'AM' if h < 12 else 'PM'
        hour_24 = f"{h:02d}:{m:02d}"
        hour_12 = f"{h%12 if h%12!=0 else 12:02d}:{m:02d} {am_pm}"
        tod = 'Madrugada' if 0 <= h < 6 else 'Mañana' if 6 <= h < 12 else 'Tarde' if 12 <= h < 18 else 'Noche'
        is_business = 8 <= h <= 18
        shift = 'Turno 1' if 0 <= h < 8 else 'Turno 2' if 8 <= h < 16 else 'Turno 3'
        dim_time.append((time_key, h, m, 0, tod, hour_24, hour_12, am_pm, is_business, shift))

insert_in_batches(cur_sf, """
INSERT INTO dim_time (time_key, hour, minute, second, time_of_day, hour_24, hour_12, am_pm, is_business_hour, shift)
VALUES (%s,%s,%s,%s,%s,%s,%s,%s,%s,%s)
""", dim_time, batch_size=5000, table_name="dim_time")
```

```
[57]
[2025-10-13 20:41:45.377847] dim_time - batch 1 success - 1440 registros
```

```
INFO:snowflake.connector.connection:Snowflake Connector for Python Version: 4.0.0, Python Version: 3.13.0, Platform: Windows-10-10.0.19043-SP0
INFO:snowflake.connector.connection:Connecting to GLOBAL Snowflake domain
C:\Users\Enzo\AppData\Local\Temp\ipykernel_19296\2529720248.py:79: UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not
  df = pd.read_sql(query, conn_pg)
INFO:__main__:📥 Datos extraídos: 125850 filas
C:\Users\Enzo\AppData\Local\Temp\ipykernel_19296\2529720248.py:104: UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are no
  df_vehicle = pd.read_sql("SELECT VEHICLE_ID, VEHICLE_KEY FROM DIM_VEHICLE", conn_sf)
C:\Users\Enzo\AppData\Local\Temp\ipykernel_19296\2529720248.py:105: UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are no
  df_driver = pd.read_sql("SELECT DRIVER_ID, DRIVER_KEY FROM DIM_DRIVER", conn_sf)
C:\Users\Enzo\AppData\Local\Temp\ipykernel_19296\2529720248.py:106: UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are no
  df_route = pd.read_sql("SELECT ROUTE_ID, ROUTE_KEY FROM DIM_ROUTE", conn_sf)
C:\Users\Enzo\AppData\Local\Temp\ipykernel_19296\2529720248.py:107: UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are no
  df_customer = pd.read_sql("SELECT CUSTOMER_NAME, CUSTOMER_KEY FROM DIM_CUSTOMER", conn_sf)
INFO:__main__:🔄 Transformando FACT_DELIVERIES
INFO:__main__:📊 FACT_DELIVERIES transformada: 126048 registros
INFO:__main__:✅ Insertadas 126048 filas en FACT_DELIVERIES
```

Finalmente verificamos que los datos estén correctamente cargados en nuestras tablas

```sql
SELECT *
FROM dim_driver
LIMIT 10;
```

Results (just now)

Table | Chart — 10 rows — 112ms

	DRIVER_KEY	DRIVER_ID	EMPLOYEE_CODE	FULL_NAME	LICENSE_NUMBER	LICENSE_EXPIRY	PHONE
1	1	1	EMP1000	Yéssica Bejarano	LIC94868	2026-06-05	+34885 26 33 07
2	2	2	EMP1001	Yaiza Sanjuan	LIC16888	2025-12-08	+34 807808496
3	3	3	EMP1002	Francisco Calleja	LIC35420	2026-07-22	+34806 14 70 53
4	4	4	EMP1003	Jose Miguel Romeu	LIC19710	2026-08-05	+34930 425 385
5	5	5	EMP1004	Paz Segovia	LIC23559	2026-04-21	+34 888 920 981
6	6	6	EMP1005	Aura Boada	LIC52267	2026-04-21	+34 941 943 617
7	7	7	EMP1006	Borja Duque	LIC13426	2026-02-25	+34 848 96 03 65
8	8	8	EMP1007	Maricruz Paz	LIC54431	2026-01-25	+34 927 38 45 46
9	9	9	EMP1008	Angelina Tomas	LIC68320	2026-10-02	+34979 803 944
10	10	10	EMP1009	Leopoldo Torrijos	LIC70144	2026-05-13	+34 924037998

```
171
172     SELECT *
173     FROM fact_deliveries
174     LIMIT 10;
175
```

Results (just now)

Table | Chart — 10 rows — 113ms

	DELIVERY_KEY	DATE_KEY	SCHEDULED_TIME_KEY	DELIVERED_TIME_KEY	VEHICLE_KEY	DRIVER_KEY	ROUTE_K
1	1	20231216	605	621	97	215	
2	2	20231216	605	621	97	715	
3	3	20231216	648	651	97	215	
4	4	20231216	648	651	97	715	
5	5	20231216	731	737	97	215	
6	6	20231216	731	737	97	715	
7	7	20231216	813	822	97	215	
8	8	20231216	813	822	97	715	
9	9	20231216	856	922	97	215	
10	10	20231216	856	922	97	715	

```
176     SELECT *
177     FROM dim_date
178     LIMIT 10;
```

Results (just now)

Table | Chart — 10 rows — 957ms

	DATE_KEY	FULL_DATE	DAY_OF_WEEK	DAY_NAME	DAY_OF_MONTH	DAY_OF_YEAR	WEEK_OF_YEAR	MON
1	20231024	2023-10-24	2	Tuesday	24	297	43	
2	20230204	2023-02-04	6	Saturday	4	35	5	
3	20241116	2024-11-16	6	Saturday	16	321	46	
4	20230303	2023-03-03	5	Friday	3	62	9	
5	20230325	2023-03-25	6	Saturday	25	84	12	
6	20241023	2024-10-23	3	Wednesday	23	297	43	
7	20230124	2023-01-24	2	Tuesday	24	24	4	
8	20230427	2023-04-27	4	Thursday	27	117	17	
9	20240104	2024-01-04	4	Thursday	4	4	1	
10	20230127	2023-01-27	5	Friday	27	27	4	

```
180     SELECT *
181     FROM dim_time
182     LIMIT 10;
183
184     SELECT *
```

Results (just now)

Table | Chart — 10 rows — 679ms

	TIME_KEY	HOU	MINUTE	SECOND	TIME_OF_DAY	HOUR_24	HOUR_12	AM_PM	IS_BUSINESS_HOU
1	0	0	0	0	Madrugada	00:00	12:00 AM	AM	FALSE
2	100	0	1	0	Madrugada	00:01	12:01 AM	AM	FALSE
3	200	0	2	0	Madrugada	00:02	12:02 AM	AM	FALSE
4	300	0	3	0	Madrugada	00:03	12:03 AM	AM	FALSE
5	400	0	4	0	Madrugada	00:04	12:04 AM	AM	FALSE
6	500	0	5	0	Madrugada	00:05	12:05 AM	AM	FALSE
7	600	0	6	0	Madrugada	00:06	12:06 AM	AM	FALSE
8	700	0	7	0	Madrugada	00:07	12:07 AM	AM	FALSE
9	800	0	8	0	Madrugada	00:08	12:08 AM	AM	FALSE

```sql
SELECT *
FROM dim_route
LIMIT 10;

SELECT *
FROM dim_vehicle
```

Results (just now)

Table | Chart — 10 rows — 593ms

	ROUTE_KEY	ROUTE_ID	ROUTE_CODE	ORIGIN_CITY	DESTINATION_CITY	DISTANCE_KM	ESTIMATED_DURA
1	1	1	R0001	Buenos Aires	Córdoba	442.99	
2	2	2	R0002	Córdoba	Rosario	226.58	
3	3	3	R0003	Bahía Blanca	La Plata	188.88	
4	4	4	R0004	Buenos Aires	Mar del Plata	569.00	
5	5	5	R0005	Salta	La Plata	373.00	
6	6	6	R0006	La Plata	Rosario	1199.26	
7	7	7	R0007	Salta	La Plata	369.13	
8	8	8	R0008	Mar del Plata	Córdoba	29.56	
9	9	9	R0009	Bahía Blanca	Mar del Plata	1086.90	
10	10	10	R0010	Salta	Rosario	771.21	

```sql
SELECT *
FROM dim_vehicle
LIMIT 10;

SELECT *
```

Results (just now)

Table | Chart — 10 rows — 340ms

	VEHICLE_KEY	VEHICLE_ID	LICENSE_PLATE	VEHICLE_TYPE	CAPACITY_KG	FUEL_TYPE	ACQUISITION_DATE
1	1	1	AL 0365 FG	Camión grande	18074.00	Nafta	2020-06-19
2	2	2	6833 RDK	Camión mediano	7143.00	Eléctrico	2019-01-14
3	3	3	0891 FVJ	Camión grande	18067.00	Eléctrico	2016-03-16
4	4	4	J 0819 WU	Camión grande	15456.00	Diesel	2019-09-25
5	5	5	8186 PTG	Camión grande	13791.00	Diesel	2017-12-11
6	6	6	CO 7874 BM	Camión grande	13628.00	Eléctrico	2016-11-16
7	7	7	A 1281 LA	Motocicleta	164.00	Eléctrico	2017-08-15
8	8	8	GI 9632 CX	Van	1653.00	Eléctrico	2019-03-15
9	9	9	PO 2544 PI	Motocicleta	121.00	Diesel	2018-02-14
10	10	10	IB 8844 KB	Camión mediano	6837.00	Diesel	2019-03-23

```sql
SELECT *
FROM dim_customer
LIMIT 10;
```

Results (just now)

Table | Chart — 10 rows — 721ms

	CUSTOMER_KEY	CUSTOMER_ID	CUSTOMER_NAME	CITY	FIRST_DELIVERY_DATE	TOTAL_DELIVERIES
1	1	1	Aarón Abraham Frutos Abad	Mar del Plata	2025-10-14	1
2	2	2	Aarón Aguilera Roig	Córdoba	2025-10-14	1
3	3	3	Aarón Agullo-Yáñez	Bahía Blanca	2025-10-14	1
4	4	4	Aarón Alba Noguera	Mar del Plata	2025-10-14	1
5	5	5	Aarón Alcalá Madrid	Bahía Blanca	2025-10-14	1
6	6	6	Aarón Alfaro Manzanares	Mar del Plata	2025-10-14	1
7	7	7	Aarón Alfaro-Mínguez	Buenos Aires	2025-10-14	1
8	8	8	Aarón Almagro Calatayud	La Plata	2025-10-14	1
9	9	9	Aarón Amigó Dalmau	Buenos Aires	2025-10-14	1
10	10	10	Aarón Amor Agustín	Bahía Blanca	2025-10-14	1

Y las vistas

```
247
248     SELECT *
249     FROM v_sales_deliveries;
250
251     SELECT *
252     FROM v_operations_deliveries;
253
```

Results (35 minutes ago)

Table | Chart — 24,886 rows — 1.3s

	FULL_DATE	CUSTOMER_NAME	ORIGIN_CITY	DESTINATION_CITY	PACKAGE_WEIGHT_KG	DELIVERY_STATUS	REVE
1	2025-08-01	Oriana del Carmona	Buenos Aires	Córdoba	5150.57	ON_TIME	
2	2025-08-01	Oriana del Carmona	Buenos Aires	Córdoba	5150.57	ON_TIME	
3	2023-03-24	Oriana del Isern	La Plata	Córdoba	4899.77	ON_TIME	
4	2023-03-24	Oriana del Isern	La Plata	Córdoba	4899.77	ON_TIME	
5	2025-09-11	Osvaldo Cañellas Alcara:	La Plata	Córdoba	5534.39	ON_TIME	
6	2025-09-11	Osvaldo Cañellas Alcara:	La Plata	Córdoba	5534.39	ON_TIME	
7	2024-08-14	Osvaldo Donaire-Saez	Mar del Plata	Córdoba	4072.32	ON_TIME	
8	2024-08-14	Osvaldo Donaire-Saez	Mar del Plata	Córdoba	4072.32	ON_TIME	

```
250
251     SELECT *
252     FROM v_operations_deliveries;
253
```

Results (just now)

Table | Chart — 4,120 rows — 2.5s

	FULL_DATE	HORA	LICENSE_PLATE	CONDUCTOR	CUSTOMER_NAME	DELIVERY_TIME_MINUTES	DELAY_MINUT
1	2024-11-23	00:03	SS 8047 JM	Francisca Espinosa	Cecilio del Camacho	24	
2	2024-11-23	00:03	SS 8047 JM	Francisca Espinosa	Cecilio del Camacho	24	
3	2023-05-15	00:00	B 4544 GO	Julio César Planas	Cirino Baños	16	
4	2023-05-15	00:00	B 4544 GO	Julio César Planas	Cirino Baños	16	
5	2024-07-23	00:16	3562 DGS	Carlota Lloret	Miriam Amaya Cortina	23	
6	2024-07-23	00:16	3562 DGS	Carlota Lloret	Miriam Amaya Cortina	23	
7	2024-05-24	00:07	0402 HSX	Amando Velázquez	Sonia Godoy Tejada	14	
8	2024-05-24	00:07	0402 HSX	Amando Velázquez	Sonia Godoy Tejada	14	

Ademas ejecutamos el script para la automatización de la ingesta diaria

```
    print(f"✅ ETL finalizado - {datetime.now()}")

# ================================================
# SCHEDULER
# ================================================
schedule.every().day.at(LOAD_HOUR).do(run_daily_etl)
print(f"⏰ Scheduler activo, ETL se ejecutará cada día a las {LOAD_HOUR}")

while True:
    schedule.run_pending()
    time.sleep(60)
```

```
⏰ Scheduler activo, ETL se ejecutará cada día a las 10:24
```

Avance 4: Arquitectura en la nube con ADW

a. Estructura general del proyecto:

El proyecto FleetLogix usa AWS para manejar datos de entregas provenientes de apps móviles de conductores. La arquitectura combina servicios serverless (sin servidores) y bases gestionadas.

El Diagrama se puede dividir en 4 capas para que se vea ordenado:

1. Capa de Entrada (Frontend / Ingreso de datos): 📱
   Aplicación móvil de conductores.

Envía eventos como:

- “Entrega completada”
- “Posición GPS actual”
- “Inicio de viaje”

Todos estos mensajes entran al sistema a través de:

- Amazon API Gateway
- Expone endpoints HTTP (por ejemplo /deliveries/verify).
- Recibe los requests y los pasa a las funciones Lambda correspondientes.

2. Capa de Procesamiento (Lógica de negocio)

AWS Lambda (3 funciones):

a. lambda_verificar_entrega: Consulta DynamoDB para ver si una entrega se completó.
   Trigger: API Gateway

b. lambda_calcular_eta: Calcula ETA usando coordenadas y velocidad.
   Trigger: EventBridge (cada 5 min)

c. lambda_alerta_desvio: Detecta desvíos de ruta y envía alertas.

Trigger: Kinesis (datos de GPS en tiempo real)

3. Capa de Almacenamiento

Amazon DynamoDB

- deliveries_status: estado de entregas
- vehicle_tracking: seguimiento y ETA
- routes_waypoints: rutas esperadas
- alerts_history: alertas generadas

Amazon S3

- Guarda los datos históricos exportados (copias de seguridad, reportes, logs, etc.)

Opcionalmente, si se agregan análisis después, se podrían incluir Snowflake o Redshift como capa analítica

4. Capa de Notificaciones y Automatización

Amazon SNS (Simple Notification Service)

- Envía alertas a supervisores o sistemas externos si hay desvíos.

Amazon EventBridge

- Programa tareas automáticas (como el cálculo ETA cada 5 minutos).

Amazon Kinesis

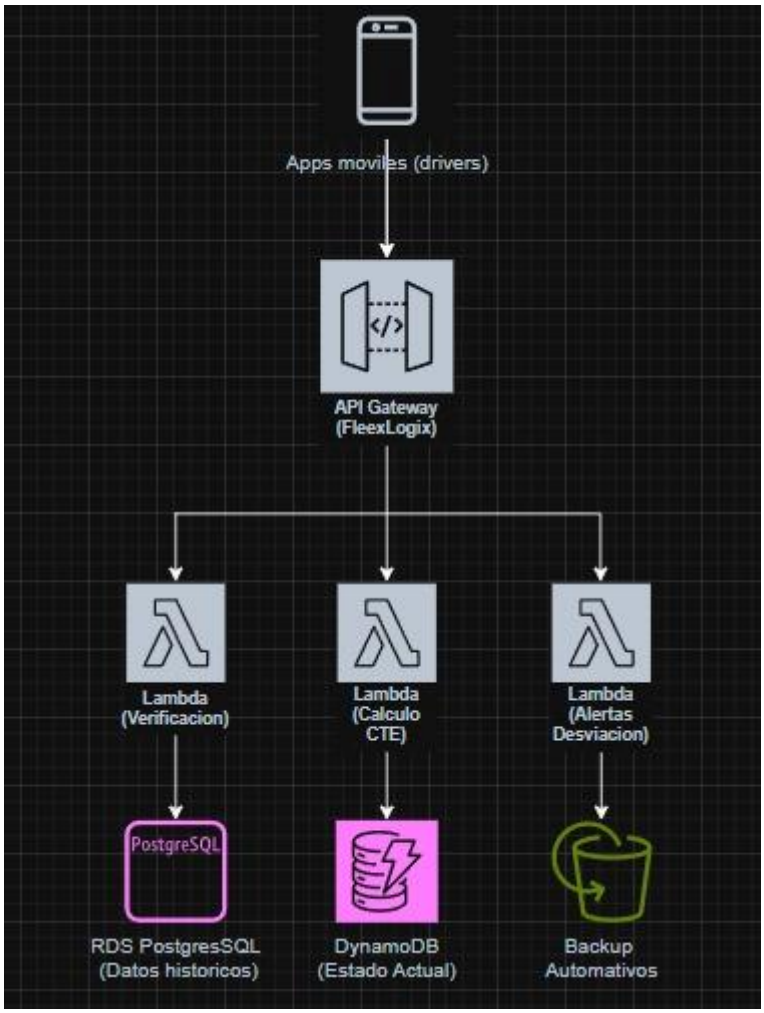
- Recibe el flujo de datos GPS en tiempo real.

Flujo de datos resumido

1) El conductor marca entrega → App móvil → API Gateway → Lambda 1 → DynamoDB
2) Cada 5 min → EventBridge ejecuta Lambda 2 → calcula ETA → guarda tracking
3) Cada actualización GPS → Kinesis → Lambda 3 → analiza desvío → SNS + DynamoDB

4) Todos los datos diarios se guardan en S3 (histórico o backups)

Diagrama:



1. Archivo aws_setup.py:

Este script automatiza la creación de los servicios base de AWS necesarios para el proyecto FleetLogix, usando la librería boto3 (el SDK oficial de AWS para Python). En otras palabras: en lugar de crear manualmente los recursos en la consola web de AWS, este script los crea programáticamente.

2. Recursos creados y función de cada uno

Recurso	Servicio AWS	Qué hace	Función en FleetLogix
RDS PostgreSQL	Amazon RDS	Crea una base de datos PostgreSQL administrada	Guarda la información transaccional y relacional (viajes, vehículos, entregas, conductores, etc.)
S3 Bucket	Amazon S3	Crea un bucket con subcarpetas y reglas de ciclo de vida	Almacena los datos históricos de las entregas, respaldos y logs
DynamoDB (4 tablas)	Amazon DynamoDB	Crea tablas NoSQL	Guarda datos operativos y en tiempo real (tracking, alertas, estados)
IAM Role para Lambda	AWS IAM	Crea un rol con permisos específicos	Permite que las funciones Lambda puedan leer/escribir en S3, DynamoDB y enviar notificaciones
Backup Snapshot RDS	Amazon RDS	Crea snapshot inicial y configura backups automáticos	Asegura disponibilidad y recuperación ante fallos
Script de migración	Shell script (migrate_to_rds.sh)	Crea un archivo para migrar datos desde una base local a RDS	Automatiza el traspaso de la base de datos local al entorno AWS

3. Explicación de funciones clave

a. **crear_rds_postgresql():** tener una base de datos relacional persistente y escalable.
    - Crea una instancia de **Amazon RDS** con el motor **PostgreSQL 15.4**, usando la clase gratuita db.t3.micro.
      Incluye configuraciones de:
    - **Backups automáticos (7 días)**
    - **Ventanas de mantenimiento**
    - **Accesibilidad pública** (para pruebas)
    - **Etiquetas (tags)** para identificar el entorno
b. **crear_s3_bucket():** Almacena de forma segura y organizada todos los datos históricos del proyecto.

Crea el bucket fleetlogix-data y dentro de él genera la siguiente estructura:

raw-data/

processed-data/

backups/

logs/

También define una política de ciclo de vida (Lifecycle Configuration): Después de 90 días, los archivos de raw-data/ se mueven automáticamente al almacenamiento Glacier, más barato, para archivo histórico.

c. **crear_tablas_dynamodb():** Crea cuatro tablas NoSQL de acceso rápido, para manejar la parte en tiempo real del sistema (tracking, alertas, actualizaciones).

Tabla	Clave primaria	Qué almacena
deliveries_status	delivery_id	Estado actual de cada entrega
vehicle_tracking	vehicle_id + timestamp	Posición geográfica del vehículo en tiempo real

routes_waypoints	route_id	Puntos de la ruta o camino a seguir
alerts_history	vehicle_id + timestamp	Historial de alertas y desviaciones

d. **configurar_backups_automaticos():** garantiza la recuperación ante desastres. Crea un snapshot inicial de la base de datos RDS y confirma la política de backups automáticos (ya definida en la función anterior).

e. **migrar_datos_postgresql():** mueve tus datos locales (por ejemplo, de localhost:5432) al nuevo entorno cloud sin perder nada.
Genera un script de bash (migrate_to_rds.sh) con los pasos para migrar una base PostgreSQL local al RDS creado.
El script:
   - Hace un pg_dump de la base local
   - Crea la base vacía en RDS
   - Restaura los datos en RDS con psql

f. **crear_rol_iam_lambda()** Permite que las funciones Lambda se integren con todos los servicios necesarios.
Crea un rol de permisos llamado **FleetLogixLambdaRole** con las siguientes políticas:
   - **AWSLambdaBasicExecutionRole**: Permite ejecutar Lambdas y escribir logs en CloudWatch
   - **AmazonDynamoDBFullAccess**: Acceso a las tablas operativas
   - **AmazonS3FullAccess**: Leer y escribir datos en el bucket.
   - **AmazonSNSFullAccess**: Enviar notificaciones (por ejemplo, alertas de desvío)

g. **main()**

Ejecuta todo el flujo completo:

1. Crea RDS
2. Crea S3
3. Crea DynamoDB
4. Configura backups
5. Crea el script de migración
6. Crea el rol IAM
7. Guarda todo en aws_config.json

Cómo se vincula esto con las funciones Lambda

Este aws_setup.py solo prepara la infraestructura.
Las funciones Lambda (por ejemplo verificar-entrega, calcular-eta, alerta-desvio) se crean aparte, y usan los recursos definidos aquí:

- Guardan entregas procesadas en DynamoDB
- Escriben registros históricos en S3
- Se autentican mediante el rol IAM creado
- Envían notificaciones o alertas mediante SNS

Funciones Lambda: Representan el "cerebro" del flujo operativo de FleetLogix en AWS.

Explicación de la función lambda_verificar_entrega:

Validar si una entrega fue completada, consultando la tabla deliveries_status en DynamoDB. Esta Lambda actúa como intermediario entre el sistema móvil y la base de datos, asegurando que las apps móviles obtengan la información correcta sin acceder directamente al almacenamiento.

Flujo paso a paso:

1. Recibe un evento desde la app móvil a través del API Gateway, con el delivery_id y opcionalmente tracking_number.
2. Valida que haya un delivery_id, si no lo hay, devuelve un error 400.
3. Consulta DynamoDB (deliveries_status) buscando esa entrega.

4. Si existe:
    - Revisa si status es "delivered".
    - Devuelve JSON con el estado actual, fecha de entrega y si está completada.

   Si no existe:
    - Devuelve error 404 (“Entrega no encontrada”).

   Si ocurre algún fallo de conexión o lectura:
    - Devuelve error 500 con el detalle del error.

Explicación de lambda_calcular_eta:

Esta Lambda se ejecuta automáticamente cada 5 minutos (EventBridge rule), ayudando a actualizar el estado en tiempo real de todos los camiones.

Flujo paso a paso:

1. Recibe desde EventBridge o una app los datos:
2. Calcula la distancia restante usando una fórmula simple de Haversine (aprox. 111 km por grado).
3. Divide la distancia por la velocidad actual: obtiene horas restantes.
4. Calcula el ETA (datetime.now() + timedelta(hours)).
5. Guarda en la tabla vehicle_tracking de DynamoDB:
    1. Ubicación actual y destino
    2. Distancia restante
    3. ETA
    4. Velocidad actual
6. Devuelve una respuesta con la distancia y ETA estimada.

Explicación de lambda_alerta_desvio: Detecta si un vehículo se desvía de su ruta predefinida y enviar una alerta.

Flujo paso a paso:

1. Recibe evento.
2. Busca en DynamoDB (routes_waypoints) los puntos de ruta esperada.
3. Calcula la distancia mínima entre el vehículo y la ruta.
4. Si se aleja más de 5 km del recorrido entonces considera que hay un desvío.

5. En caso de desvío:
    - Envía un mensaje al SNS topic fleetlogix-alerts.
    - Guarda el evento en la tabla alerts_history para registro.
6. Devuelve si hubo o no desvío, y a qué distancia.

Extracredit:

El script sirve para predecir el consumo de combustible de un viaje, usando datos históricos de entregas, vehículos y rutas. Esto ayuda a FleetLogix a planificar mejor los viajes, optimizar costos de combustible y mejorar la eficiencia operativa.

- Se obtiene un dataset con las variables más relevantes para predecir el consumo:
  - distance_km: distancia del viaje
  - delivery_time_minutes: tiempo que tardó la entrega
  - vehicle_type: tipo de vehículo
  - capacity_kg: capacidad del vehículo
  - age_months: antigüedad del vehículo
  - estimated_duration_hours: duración estimada de la ruta
  - fuel_consumed_liters: combustible realmente consumido (target)
- Se hace un **join con las tablas de vehículos y rutas** para tener toda la información necesaria.

En este caso, el resultado arroja que para un viaje dado donde: nuevo_viaje =

'distance_km': [120],

'delivery_time_minutes': [150],

'vehicle_type': ['Camión mediano'],

'capacity_kg': [3000],

'age_months': [36],

'estimated_duration_hours': [2.5]

se consumiran 19.36 litros

```
#------ EXTRACREDIT ------

### El script sirve para predecir el consumo de combustible de un viaje, usando datos históricos de entregas, vehículos y rutas.
# Esto ayuda a FleetLogix a planificar mejor los viajes, optimizar costos de combustible y mejorar la eficiencia operativa.

import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error, r2_score
import snowflake.connector

# ================================
# 1 Conexión a Snowflake
# ================================
conn = snowflake.connector.connect(
    user='ENZOZAMBON',
    password='AdiraelSpriest12345',
    account='GTUWIRG-PU45327',
    warehouse='FLEETLOGIX_WH',
    database='FleetLogix_dhw',
    schema='ANALYTICS'
)

# ================================
# 2 Consulta de datos relevantes
# ================================
query = """
SELECT
    F.DISTANCE_KM,
    F.DELIVERY_TIME_MINUTES,
    V.VEHICLE_TYPE,
    V.CAPACITY_KG,
    V.AGE_MONTHS,
    R.ESTIMATED_DURATION_HOURS,
    F.FUEL_CONSUMED_LITERS
FROM FACT_DELIVERIES F
JOIN DIM_VEHICLE V ON F.VEHICLE_KEY = V.VEHICLE_KEY
JOIN DIM_ROUTE R ON F.ROUTE_KEY = R.ROUTE_KEY
WHERE F.FUEL_CONSUMED_LITERS IS NOT NULL
"""
df = pd.read_sql(query, conn)

# Convertir columnas a minúsculas para trabajar más cómodo en pandas
df.columns = df.columns.str.lower()
```

```
# ================================
# 3 Preparación de datos
# ================================
X = df.drop(columns=['fuel_consumed_liters'])
y = df['fuel_consumed_liters']

categorical_features = ['vehicle_type']
numerical_features = ['distance_km', 'delivery_time_minutes', 'capacity_kg', 'age_months', 'estimated_duration_hours']

preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_features),
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features)
    ]
)

# ================================
# 4 División de datos en train/test
# ================================
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# ================================
# 5 Pipeline + Modelo
# ================================
model = Pipeline([
    ('preprocessor', preprocessor),
    ('regressor', RandomForestRegressor(n_estimators=200, max_depth=10, random_state=42))
])

model.fit(X_train, y_train)
y_pred = model.predict(X_test)

# ================================
# 6 Evaluación
# ================================
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f"Mean Absolute Error: {mae:.2f} litros")
print(f"R² Score: {r2:.2f}")

# ================================
# 7 Predicción de un nuevo viaje
# ================================
nuevo_viaje = pd.DataFrame({
    'distance_km': [120],
    'delivery_time_minutes': [150],
    'vehicle_type': ['Camión mediano'],
    'capacity_kg': [3000],
    'age_months': [36],
    'estimated_duration_hours': [2.5]
})

consumo_estimado = model.predict(nuevo_viaje)
print(f"Consumo estimado de combustible: {consumo_estimado[0]:.2f} litros")
```

✓ 1m 14.3s

```
C:\Users\Enzo\AppData\Local\Temp\ipykernel_12708\2578783964.py:40: UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or
  df = pd.read_sql(query, conn)
Mean Absolute Error: 19.48 litros
R² Score: 0.91
Consumo estimado de combustible: 19.36 litros
```