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.

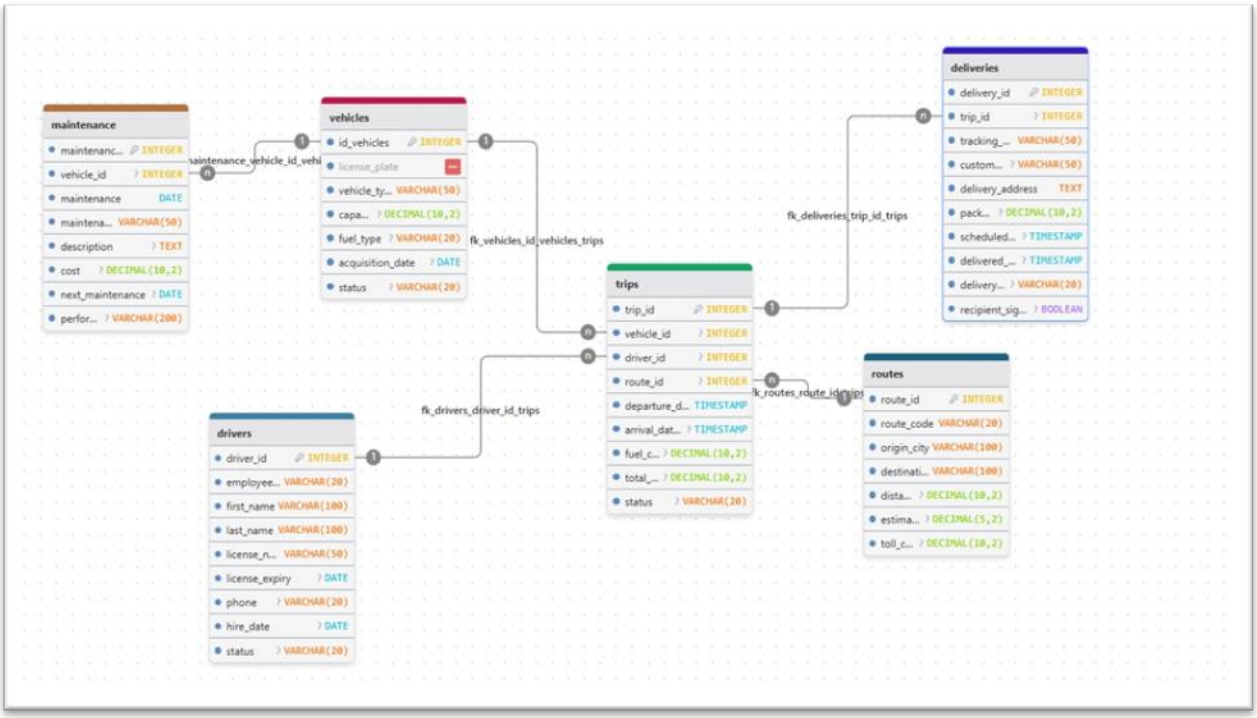
     Relaciones:

     - 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:

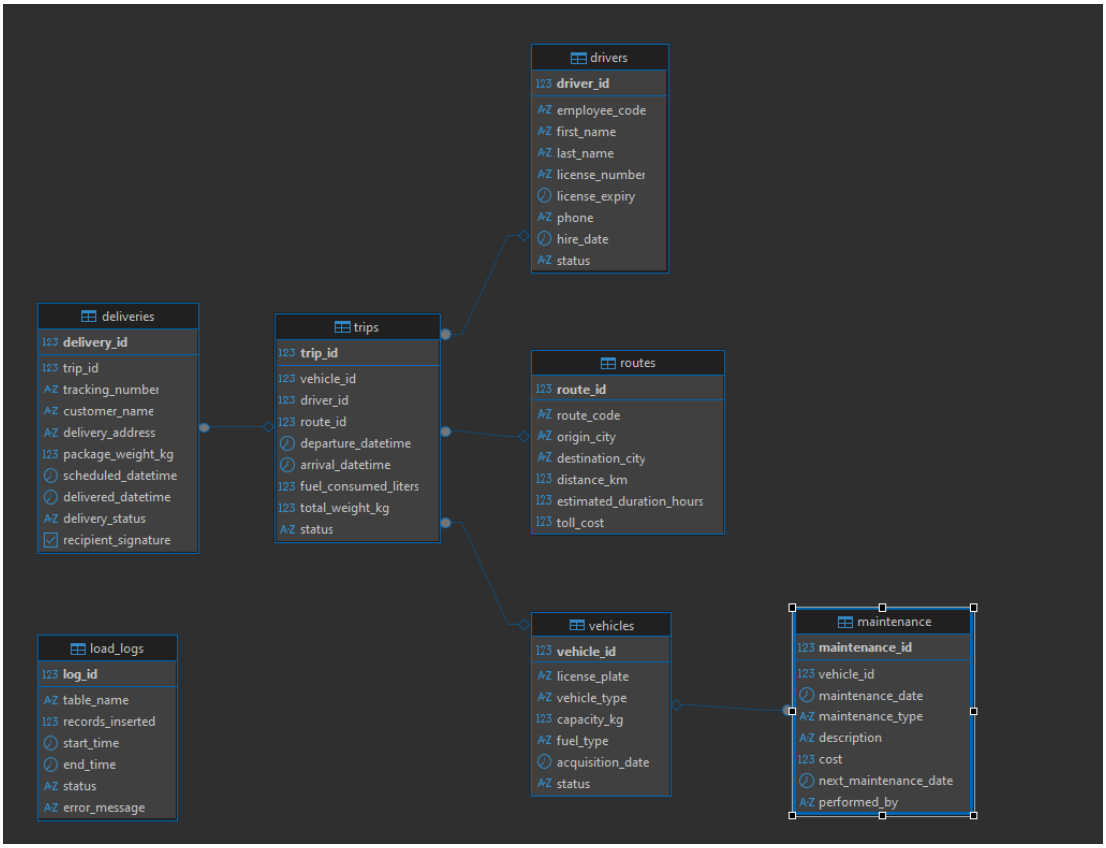
```
-- ============================================================
-- FLEETLOGIX DATABASE SETUP
-- Sistema de Gestión de Transporte y Logística
-- ============================================================

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 | ecord

	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:

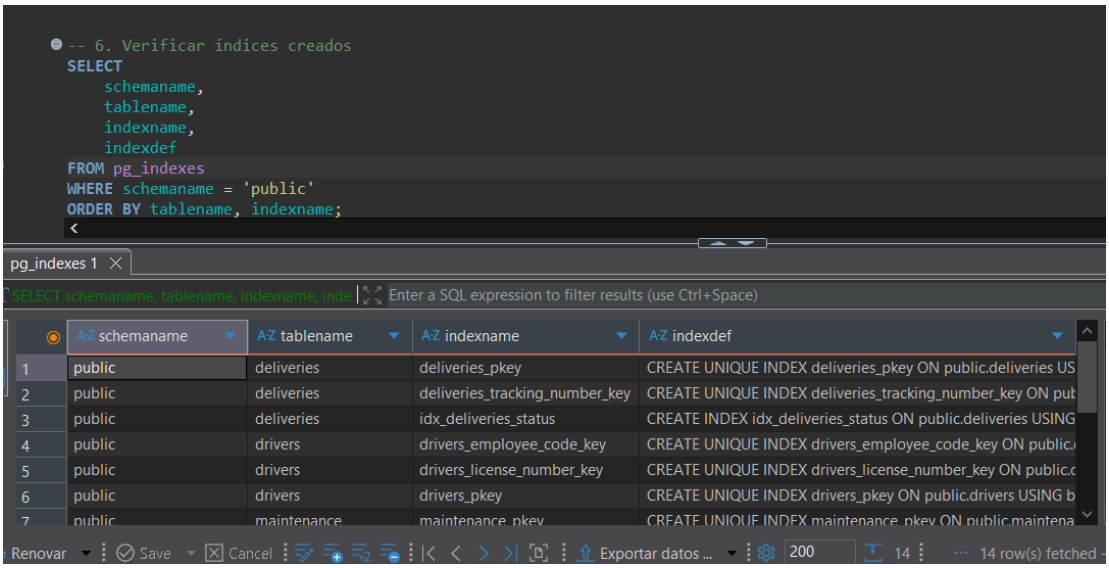
```sql
-- 5. Verificar las relaciones (foreign keys)
SELECT
    tc.table_name AS tabla_origen,
    kcu.column_name AS columna_origen,
    ccu.table_name AS tabla_referencia,
    ccu.column_name AS columna_referencia
FROM information_schema.table_constraints AS tc
JOIN information_schema.key_column_usage 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

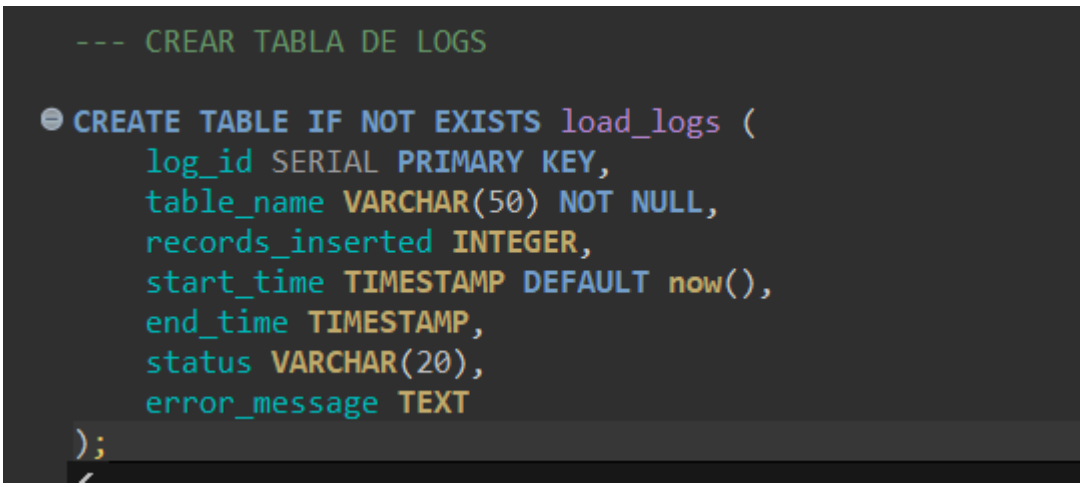
SELECT tc.table_name AS tabla_origen, kcu.column | Enter a SQL expression to filter results (use Ctrl+Space)

	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:



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



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

```python
# ---------- # 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:

```python
# ---------- 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()
```

```
    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
```

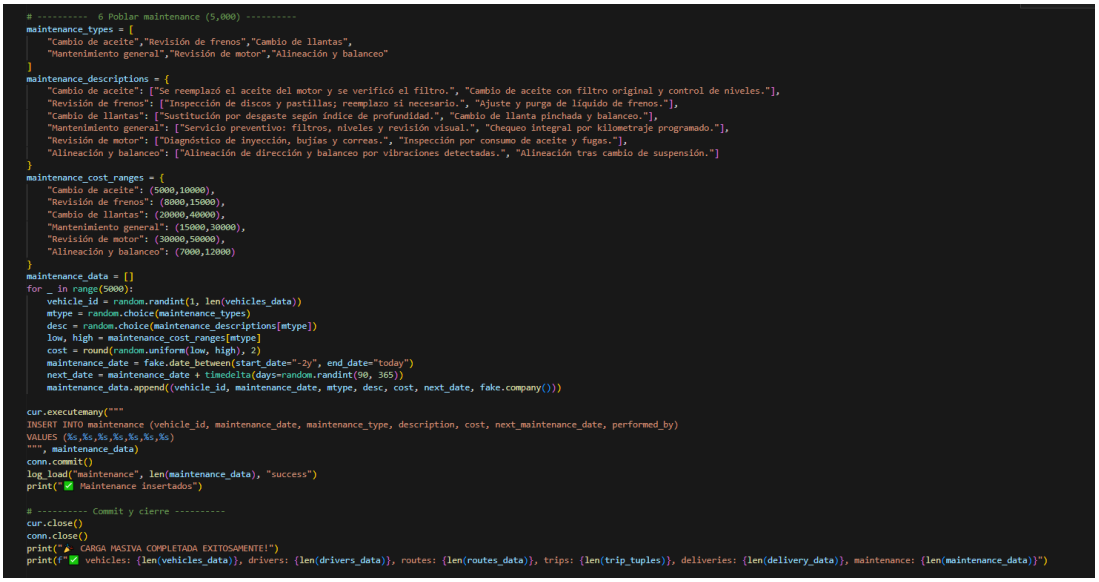
```
total_weight = []
fuel_consumed = []
final_arrivals = []
for i, vid in enumerate(chosen_vehicles):
    cap = vehicle_id_to_capacity.get(int(vid), 1000)
    v_status = vehicle_id_to_status.get(int(vid), "active")
    if v_status == "active":
        w = round(random.uniform(50, cap), 2)
    else:
        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

SELECT 'vehicles' AS table_name, COUNT(*) AS tota | 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

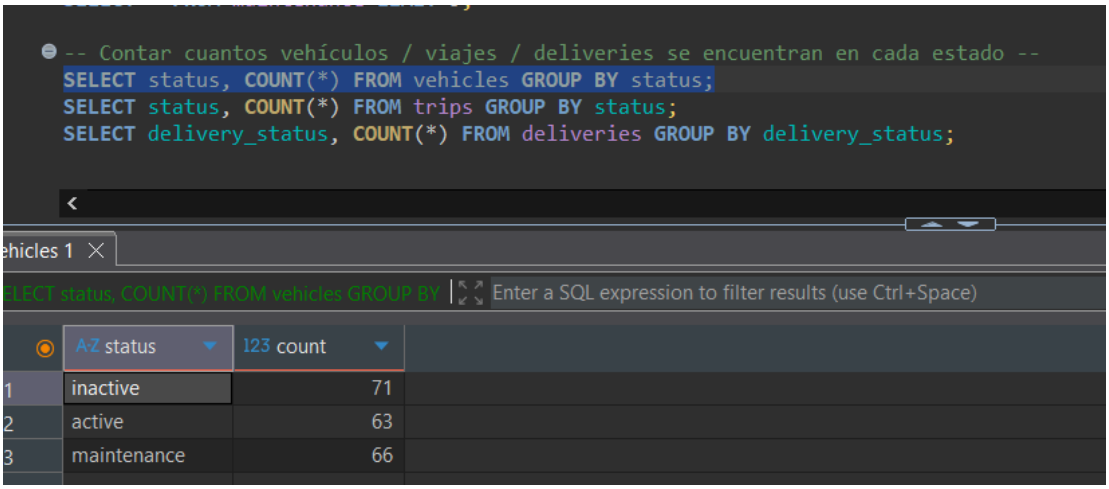
```sql
-- Podemos ver los primeros registros de cada tabla --
SELECT * FROM vehicles LIMIT 5;
SELECT * FROM drivers LIMIT 5;
SELECT * FROM routes LIMIT 5;
SELECT * FROM trips LIMIT 5;
SELECT * FROM deliveries LIMIT 5;
SELECT * FROM maintenance LIMIT 5;
```

vehicles 1

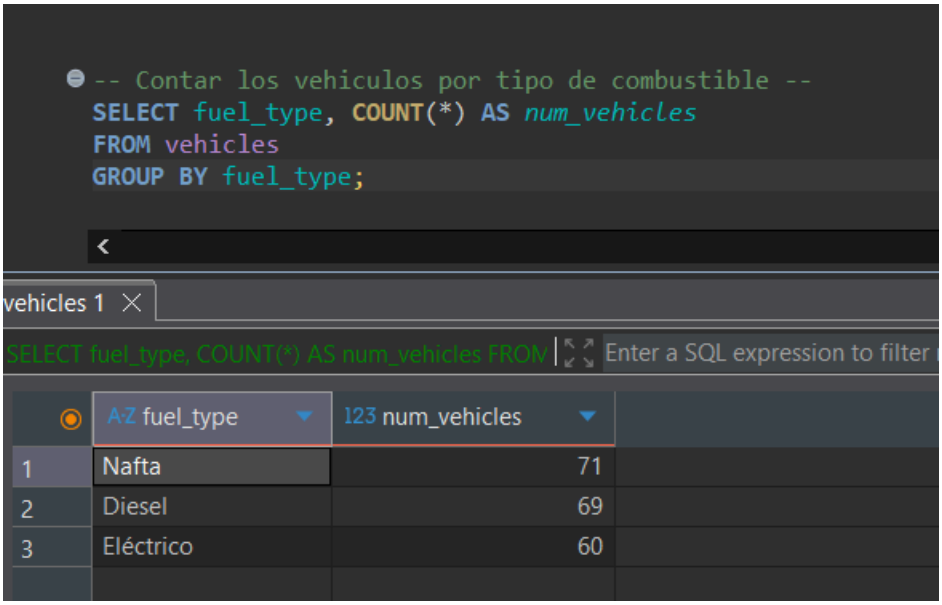
SELECT * FROM vehicles LIMIT 5 | Enter a SQL expression to filter results (use Ctrl+Space)

	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:



Contar los vehículos por tipo de combustible:



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:

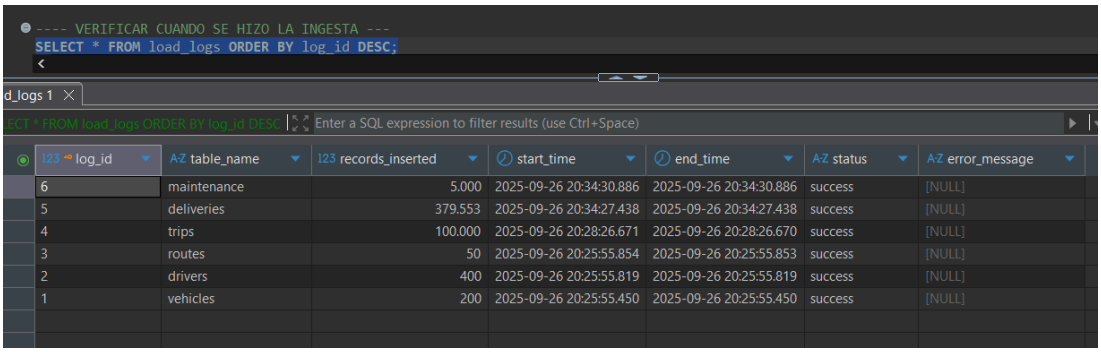
La siguiente consulta devuelve todos los viajes donde la llegada es antes o igual a la salida. Si no devuelve filas entonces todos los viajes son consistentes.

```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]