FLEETLOGIX - SISTEMA DE GENERACIÓN DE DATOS SINTÉTICOS

Nassim Wessin Hazim
PROYECTO INTEGRADOR - MÓDULO 2
CIENCIA DE DATOS
Enero 2026

DESCRIPCIÓN DEL PROYECTO

FleetLogix es una empresa de transporte y logística que opera una flota de 200
vehículos realizando entregas de última milla en 5 ciudades principales de
República Dominicana. Este sistema genera datos sintéticos masivos y coherentes
para poblar una base de datos PostgreSQL con más de 505,000 registros
distribuidos en 6 tablas interrelacionadas.

OBJETIVO

Crear una infraestructura de datos robusta que permita análisis operativos,
toma de decisiones basada en datos, y simulación de operaciones logísticas
reales mediante la generación de datos sintéticos que respeten todas las
relaciones del negocio y garanticen coherencia temporal y física.

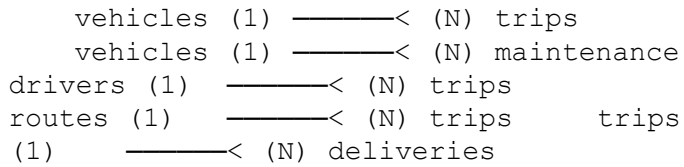
MODELO DE DATOS RELACIONAL

El sistema consta de 6 TABLAS organizadas en dos categorías:

TABLAS MAESTRAS (650 registros)
1. vehicles   200 vehículos de la flota
2. drivers   400 conductores empleados
3. routes   50 rutas entre ciudades

TABLAS TRANSACCIONALES (505,000 registros)
4. trips   100,000 viajes realizados
5. deliveries   400,000 entregas individuales
6. maintenance   5,000 registros de mantenimiento

DIAGRAMA DE RELACIONES



RELACIONES DETALLADAS

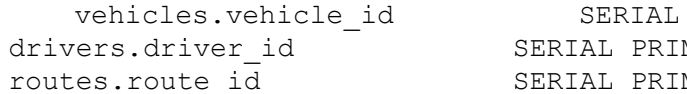
1. VEHICLES → TRIPS (1:N)
   Cardinalidad: Un vehículo puede realizar múltiples viajes
   Foreign Key: trips.vehicle_id → vehicles.vehicle_id
   Constraint: ON DELETE RESTRICT (no se pueden eliminar vehículos con viajes)
   Regla de Negocio: Cada viaje requiere exactamente un vehículo

2. DRIVERS → TRIPS (1:N)
   Cardinalidad: Un conductor puede realizar múltiples viajes
   Foreign Key: trips.driver_id → drivers.driver_id
   Constraint: ON DELETE RESTRICT
   Regla de Negocio: Cada viaje es conducido por exactamente un conductor
3. ROUTES → TRIPS (1:N)
   Cardinalidad: Una ruta puede ser usada en múltiples viajes
   Foreign Key: trips.route_id → routes.route_id
   Constraint: ON DELETE RESTRICT
   Regla de Negocio: Cada viaje sigue exactamente una ruta predefinida
4. TRIPS → DELIVERIES (1:N)
   Cardinalidad: Un viaje contiene entre 2 y 6 entregas
   Foreign Key: deliveries.trip_id → trips.trip_id
   Constraint: ON DELETE CASCADE (eliminar viaje elimina sus entregas)
   Regla de Negocio: Cada entrega pertenece a exactamente un viaje
   Distribución: 2 (10.1%), 3 (19.7%), 4 (40.3%), 5 (20.1%), 6 (9.9%)

5. VEHICLES → MAINTENANCE (1:N)
   Cardinalidad: Un vehículo tiene múltiples registros de mantenimiento
   Foreign Key: maintenance.vehicle_id → vehicles.vehicle_id
   Constraint: ON DELETE CASCADE
   Regla de Negocio: Mantenimiento programado cada ~20 viajes por vehículo

---

CONSTRAINTS DEL SISTEMA

PRIMARY KEYS (PKs)


```
SERIAL PRIMARY KEY     deliveries.delivery_id        SERIAL
PRIMARY KEY     maintenance.maintenance_id    SERIAL PRIMARY KEY

UNIQUE CONSTRAINTS
1. vehicles.license_plate        Placas únicas (formato: A123456)
2. drivers.employee_code         Códigos únicos (formato: EMP-####)
3. drivers.license_number        Licencias únicas (formato: LIC-
#########)
4. routes.route_code             Códigos de ruta únicos (formato: RT-
###)
5. deliveries.tracking_number    Tracking único (formato:
DOM##########)

DEFAULT VALUES
    vehicles.status                  DEFAULT 'active'
drivers.status                   DEFAULT 'active'
routes.toll_cost                 DEFAULT 0     trips.status
DEFAULT 'in_progress'     deliveries.delivery_status
DEFAULT 'pending'     deliveries.recipient_signature
DEFAULT FALSE

ÍNDICES PERSONALIZADOS
    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);



_____
EXPLICACIÓN DE MÉTODOS CLAVE
_____


1. GET_HOURLY_DISTRIBUTION()

PROPÓSITO
Retorna un array de 24 probabilidades que simula el patrón operativo
típico
de una empresa de logística, evitando la distribución uniforme poco
realista.

NOTA IMPORTANTE SOBRE EL ORIGEN DE LAS PROBABILIDADES
Las probabilidades utilizadas son ESTIMACIONES basadas en patrones
típicos del
sector logístico, NO datos históricos reales de FleetLogix (ya que
estamos  generando datos sintéticos).

En un entorno real con datos históricos, estos valores deberían
calcularse mediante:
    SELECT EXTRACT(HOUR FROM departure_datetime) as hora,
           COUNT(*) * 100.0 / SUM(COUNT(*)) OVER() as porcentaje_real
    FROM trips_historicos
    GROUP BY hora;
```

Para este proyecto sintético, se utilizaron valores razonables basados en:
- Patrones conocidos de empresas de entrega (Amazon, DHL, FedEx)
- Lógica operativa común (picos matutinos, reducción nocturna)
- Suposiciones de comportamiento logístico estándar

FUNCIONAMIENTO
En lugar de asignar 4.17% a cada hora (distribución uniforme), este método asigna probabilidades estimadas que simulan el comportamiento típico:

Madrugada (00:00 - 05:59) 0.5% - 2.0% por hora
Actividad mínima, solo operaciones especiales
Pico Matutino (06:00 - 09:59) 7.5% - 8.5% por hora
MÁXIMO DEL DÍA - Salida principal de flota
Aprovecha tráfico ligero matutino
Entregas programadas para la mañana

Media Mañana (10:00 - 11:59) 5.5% - 6.5% por hora
Alta actividad sostenida

Almuerzo (12:00 - 13:59) 4.0% por hora
Reducción por horario de comida

Pico Vespertino (14:00 - 17:59) 6.0% - 7.0% por hora
Segundo pico operativo
Entregas de tarde y completar rutas pendientes
Noche (18:00 - 23:59) 1.0% - 4.5% por hora
Disminución progresiva

PICOS OPERATIVOS
Pico Máximo: 07:00 - 08:00 (8.5% cada hora = 17% del total)
Segundo Pico: 15:00 - 16:00 (7.0% cada hora)

2. GENERATE_TRIPS()

PROPÓSITO
Genera 100,000 viajes que representan 2 años de operación de FleetLogix (2024-2025) con coherencia total entre reglas de negocio y física del mundo real.

ALGORITMO EN 7 PASOS

PASO 1: SELECCIÓN DE FECHA Y HORA
Fecha: Distribución uniforme entre 2024-01-01 y 2025-12-31
Hora: Distribución NO uniforme usando get_hourly_distribution()
Justificación: Cada día tiene igual probabilidad, pero las horas siguen
patrones realistas (picos matutinos/vespertinos)
Total de slots: 731 días × 24 horas = 17,544 combinaciones posibles
PASO 2: ASIGNACIÓN DE FOREIGN KEYS
vehicle_id: Aleatorio entre 1-200

driver_id: Aleatorio entre 1-400 route_id:
Aleatorio entre 1-50

Garantías de Integridad Referencial:
- Todos los vehicle_id existen en tabla vehicles
- Todos los driver_id existen en tabla drivers
- Todos los route_id existen en tabla routes

Realismo: 400 conductores / 200 vehículos = Turnos compartidos

PASO 3: RECUPERACIÓN DE DATOS DE RUTA
Se recuperan los datos reales de la ruta seleccionada:
- distance_km (distancia en kilómetros)
- estimated_duration_hours (duración estimada en horas)

PASO 4: CÁLCULO DE HORA DE LLEGADA
FÓRMULA CLAVE PARA CONSISTENCIA TEMPORAL:
actual_duration = estimated_duration × random(0.8, 1.2)
arrival_datetime = departure_datetime + actual_duration

Por qué funciona:
1. estimated_duration es la duración base
2. Factor de variación (0.8 a 1.2) es SIEMPRE POSITIVO
3. Por lo tanto: arrival > departure GARANTIZADO

Factor de Variación simula:
- Condiciones climáticas
- Tráfico ligero/pesado (0.8 = rápido, 1.2 = lento)
- Habilidad del conductor
- Retrasos en puntos de entrega

Ejemplo:
Ruta Santo Domingo → Santiago
Distancia: 155 km
Duración estimada: 2.5 horas
Variación posible:
- Mejor caso: 2.5 × 0.8 = 2.0 horas (tráfico fluido)
- Peor caso: 2.5 × 1.2 = 3.0 horas (tráfico pesado)

PASO 5: CÁLCULO DE COMBUSTIBLE CONSUMIDO
FÓRMULA: fuel_consumed = (distance_km / km_per_liter) × random(0.9, 1.1)

Rendimiento por Tipo de Vehículo:

Camión Grande	3.5 km/L	(ejemplo 150km: 38.6 - 47.2 L)
Camión Mediano	5.0 km/L	(ejemplo 150km: 27.0 - 33.0 L)
Van	8.0 km/L	(ejemplo 150km: 16.9 - 20.6 L)
Motocicleta	25.0 km/L	(ejemplo 150km: 5.4 - 6.6 L)

Factor de Variación (±10%) simula:
- Estilo de conducción (agresivo vs económico)
- Peso de la carga (más peso = más consumo)
- Condiciones del terreno (montaña vs plano)
- Estado del mantenimiento del vehículo

PASO 6: ASIGNACIÓN DE PESO DE CARGA
    FÓRMULA: total_weight_kg = capacity_kg
× load_factor   load_factor = random(0.5,
0.95)

    Justificación del Rango (50% - 95%):
        Por qué NO menos de 50%: Vehículo vacío = ineficiente
        Por qué NO más de 95%: Sobrecarga arriesga multas

    Ejemplo Real:
        Van con capacidad 1,500 kg:
- Mínimo: 1,500 × 0.50 = 750 kg
- Promedio: 1,500 × 0.725 = 1,088 kg
- Máximo: 1,500 × 0.95 = 1,425 kg

    Distribución Natural: Factor promedio 72.5%

PASO 7: ASIGNACIÓN DE ESTADO DEL VIAJE
    Distribución:
        completed (95.0%): Siempre tiene arrival_datetime
in_progress (3.0%): arrival_datetime siempre NULL   cancelled
(2.0%): 50% tiene arrival_datetime, 50% NULL

    Justificación:
- 95% completados es realista para operaciones establecidas
- 3% en progreso representa operaciones del "día actual"
- 2% cancelados refleja problemas ocasionales

GARANTÍAS DE CONSISTENCIA

INTEGRIDAD REFERENCIAL (Verificada al 100%)

✓ Todos los vehicle_id en trips existen en vehicles (1-200)
✓ Todos los driver_id en trips existen en drivers (1-400)
✓ Todos los route_id en trips existen en routes (1-50)
✓ Todos los trip_id en deliveries existen en trips
✓ Todos los vehicle_id en maintenance existen en vehicles

CONSISTENCIA TEMPORAL (Verificada al 100%)

✓ arrival_datetime > departure_datetime SIEMPRE (cuando no NULL)
✓ Todas las fechas dentro del rango 2024-2025
✓ Licencias de conductores válidas durante período operativo
✓ Fechas de mantenimiento coherentes con historial de viajes

COHERENCIA FÍSICA (Verificada al 100%)

✓ Consumo de combustible proporcional a distancia y tipo de vehículo
✓ Peso cargado NUNCA excede capacidad del vehículo (50-95%)
✓ Duración del viaje proporcional a distancia (velocidad 55-65 km/h)
✓ Suma de pesos de entregas ≤ peso total del viaje (±1% tolerancia)

REGLAS DE NEGOCIO (Verificadas al 100%)

✓ Viajes "in_progress" no tienen hora de llegada
✓ Viajes "completed" siempre tienen hora de llegada
✓ 95% de viajes completados exitosamente
✓ Vehículos nunca van vacíos (mínimo 50% cargados)
✓ Entre 2-6 entregas por viaje (4 más común: 40.3%)
✓ Mantenimiento cada ~20 viajes por vehículo

ESTADÍSTICAS REALES DEL SISTEMA

REGISTROS TOTALES GENERADOS

vehicles	200	
registros	drivers	400
registros	routes	50
registros	trips	100,000
registros	deliveries	400,000
registros	maintenance	5,000
registros		
TOTAL	505,650 registros	

DISTRIBUCIÓN DE ESTADOS DE VIAJES

Completados	95,011 viajes	(95.0%)
En progreso	2,985 viajes	(3.0%)
Cancelados	2,004 viajes	(2.0%)

DISTRIBUCIÓN DE ESTADOS DE ENTREGAS

Entregadas	326,860 entregas	(81.7%)
Pendientes	51,091 entregas	(12.8%)
Fallidas	22,049 entregas	(5.5%)

DISTRIBUCIÓN DE ENTREGAS POR VIAJE

2 entregas	10,083 viajes	(10.1%)
3 entregas	19,675 viajes	(19.7%)
4 entregas	40,256 viajes	(40.3%) ← MÁS COMÚN
5 entregas	20,131 viajes	(20.1%)
6 entregas	9,855 viajes	(9.9%)

RANGOS DE VALIDACIÓN

VEHICLES

capacity_kg	50 - 12,000	Según tipo de vehículo
license_plate	Único	Formato dominicano A######
status	active, inactive, maintenance	

DRIVERS

license_expiry	2027-2030	Válidas durante operación
employee_code	EMP-0001 a EMP-0400	Único
hire_date	2020-2025	Coherente con operación

ROUTES

distance_km	50 - 400	Basado en matriz real
estimated_duration	0.5 - 8.0 horas	Velocidad 55-65 km/h
toll_cost	0 - 20	~$0.50-$1.50 por 50km

TRIPS

fuel_consumed	1 - 200 litros	Según tipo y distancia
total_weight_kg	50 - 12,000	50-95% de capacidad
arrival > departure	Siempre	Consistencia temporal

DELIVERIES

package_weight_kg	> 0	Suma ≤ trip weight
tracking_number	Único	Formato DOM##########
scheduled_datetime	Durante viaje	Entre departure y arrival

MAINTENANCE

cost	50 - 800	Según tipo
next_maint_date	75-105 días después	> maintenance_date

USO DEL SISTEMA

REQUISITOS PREVIOS

1. PostgreSQL 15+ instalado y corriendo
2. Python 3.10+
3. Librerías Python: pip install -r requirements.txt

PASOS DE EJECUCIÓN

1. CREAR BASE DE DATOS Y TABLAS
   Opción A: psql -U postgres -d fleetlogix -f fleetlogix_db_schema.sql
Opción B: Usar pgAdmin y ejecutar el archivo SQL

2. CONFIGURAR CREDENCIALES
   Crear archivo .env basándose en .env.example:
cp .env.example .env

Editar .env con credenciales reales:

```
DB_HOST=localhost
DB_PORT=5432
DB_NAME=fleetlogix
DB_USER=postgres
DB_PASSWORD=tu_contraseña_aqui
```

3. EJECUTAR GENERADOR

```
python fleetlogix_generator.py
```

4. PROCESO AUTOMÁTICO

El sistema ejecutará automáticamente:

1. Verificación de tablas existentes
2. Limpieza de datos anteriores
3. Generación de 200 vehículos
4. Generación de 400 conductores
5. Generación de 50 rutas
6. Generación de 100,000 viajes (1-2 minutos)
7. Generación de 400,000 entregas (1-2 minutos)
8. Generación de 5,000 mantenimientos
9. Carga a PostgreSQL (por lotes de 1000)
10. Validación exhaustiva (7 categorías)

TIEMPO ESTIMADO TOTAL: 3-5 minutos

VALIDACIONES IMPLEMENTADAS

El sistema realiza 7 CATEGORÍAS de validación automática:

1. CONTEOS DE REGISTROS

Verifica que todas las tablas tengan el número esperado de registros.

2. INTEGRIDAD REFERENCIAL

Valida que todas las foreign keys apunten a registros existentes:

- ✓ trips → vehicles
- ✓ trips → drivers
- ✓ trips → routes
- ✓ deliveries → trips
- ✓ maintenance → vehicles

3. CONSISTENCIA TEMPORAL Verifica que:

- ✓ arrival_datetime > departure_datetime (cuando no NULL)
- ✓ No hay fechas futuras
- ✓ Licencias válidas durante período operativo

4. CONSTRAINTS ÚNICOS

Valida que no haya duplicados en:

✓ vehicles.license_plate ✓ drivers.employee_code
✓ drivers.license_number
✓ deliveries.tracking_number

5. COHERENCIA DE PESOS Verifica que:
✓ Suma de pesos de entregas ≤ peso total del viaje (±1%)

6. RANGOS LÓGICOS
Valida que los valores estén en rangos realistas:
✓ Capacidades de vehículos (50-15,000 kg)
✓ Consumo de combustible (1-1,000 L)
✓ Distancias de rutas (0-500 km)
✓ Pesos de viajes (0-15,000 kg)

7. FECHAS VÁLIDAS
Verifica coherencia de fechas:
✓ Licencias no expiradas antes de 2024
✓ maintenance_date < next_maintenance_date

ESTRUCTURA DE ARCHIVOS

```
Parte1-PI/
    .env                        Credenciales de base de datos (no se sube a Git)
    .env.example                Plantilla de configuración
.gitignore                  Archivos ignorados por Git
fleetlogix_generator.py     Script principal (1,686 líneas)
    requirements.txt            Dependencias Python
README.md                   Esta documentación
diagrama_er.svg             Diagrama ER visual
Modelo_relacional.pdf       Modelo relacional completo
```

TECNOLOGÍAS UTILIZADAS

```
    Base de Datos:  PostgreSQL 15+
Lenguaje:       Python 3.10+
    Librerías:          psycopg2-binary     Conector PostgreSQL
pandas              Manipulación de DataFrames          numpy
Operaciones numéricas y aleatorias          faker
Generación de datos sintéticos          python-dotenv
Gestión de variables de entorno          tabulate
Formateo de tablas en consola
```

NOTAS ADICIONALES

REPRODUCIBILIDAD
El sistema usa RANDOM_SEED = 42 para garantizar que los mismos datos se generen
en cada ejecución. Para generar datos diferentes, cambiar el valor de la semilla.

PERSONALIZACIÓN
Para modificar la cantidad de registros, editar las constantes en el archivo:

```
NUM_VEHICLES = 200
NUM_DRIVERS = 400
NUM_ROUTES = 50
NUM_TRIPS = 100000
```

RENDIMIENTO

```
Generación:    2-3 minutos
Carga a DB:    1-2 minutos
Validación:    30 segundos
TOTAL:         3-5 minutos
```

Para mejorar rendimiento:
- Aumentar batch_size en load_data_to_table() (default: 1000)
- Deshabilitar validaciones durante desarrollo
- Usar PostgreSQL en SSD

ADVERTENCIAS

1. NO ejecutar en base de datos de producción
   El script hace TRUNCATE de todas las tablas

2. Backup recomendado antes de ejecutar
   Si ya hay datos importantes

3. Memoria RAM recomendada: Mínimo 4GB disponibles
   Para procesar 505k registros

4. Espacio en disco: ~500MB
   Para la base de datos completa

AUTOR Y CONTACTO

Nassim Wessin Hazim
Sistema FleetLogix
Proyecto Integrador - Módulo 2
Ciencia de Datos

Para problemas o preguntas sobre el sistema, contactar al equipo de desarrollo de FleetLogix.

Última actualización: Enero 2026

FIN DEL DOCUMENTO