MANUAL CONSULTAS SQL - AVANCE 2

Proyecto: FleetLogix – Sistema de Gestión de Transporte y Logística
Autor: Cristian García
Repositorio: https://github.com/cfgarciac/Module-2
Versión del documento: v1.0
Fecha: 12/11/2025

1. OBJETIVO Y ALCANCE

Objetivo. Transformar los registros del sistema operacional de FleetLogix en información accionable, ejecutando y documentando 12 consultas SQL (de las cuales se analizarán y presentarán 8 de forma prioritaria), con:

- Análisis de planes de ejecución (EXPLAIN ANALYZE).
- Optimización de performance mediante índices específicos.
- Medición comparativa antes y después de optimizar.
- Interpretación de negocio para cada consulta.

Alcance.

- Base de datos transaccional poblada (Avance 1).
- 12 consultas clasificadas por complejidad (3 básicas, 5 intermedias, 4 complejas).
- Al menos 5 índices diseñados y aplicados a partir de los cuellos de botella detectados.
- Evidencia de mejora (objetivo: ≥20% por consulta; aspiracional: ≥50% en las más costosas).

Fuera de alcance.

- Cambios de esquema mayores.
- ETL o Data Warehouse (cubiertos en avances posteriores).

2. CONTEXTO DE NEGOCIO

FleetLogix necesita responder preguntas clave sobre vehículos, conductores, rutas, viajes y entregas: estado operativo, puntualidad, productividad por ruta, eficiencia por tipo de vehículo y conductor, y costos de mantenimiento.
Las consultas seleccionadas permiten:

- Monitorear cumplimiento operativo (viajes y entregas a tiempo).
- Detectar riesgos (licencias por vencerse, outliers de consumo o costos).
- Optimizar eficiencia (entregas/hora, combustible por km, ranking de conductores/rutas).

3. MODELO LÓGICO (RESUMEN)

Tablas principales y llaves relevantes:

- vehicles(vehicle_id PK), attrs: license_plate, vehicle_type, capacity_kg, fuel_type, status, …
- drivers(driver_id PK), attrs: first_name, last_name, license_number, license_expiry, status, …
- routes(route_id PK), attrs: route_code, origin_city, destination_city, distance_km, …
- trips(trip_id PK, vehicle_id FK, driver_id FK, route_id FK), attrs: departure_datetime, arrival_datetime, fuel_consumed_liters, total_weight_kg, status, …
- deliveries(delivery_id PK, trip_id FK), attrs: scheduled_datetime, delivered_datetime, delivery_status, …
- maintenance(maintenance_id PK, vehicle_id FK), attrs: maintenance_date, maintenance_type, cost, …

Relaciones clave:

- trips enlaza vehículo + conductor + ruta.
- deliveries depende de trips.
- maintenance depende de vehicles.

(El ERD detallado se documentó en Avance 1; aquí basta con el recordatorio funcional para interpretar las consultas.).

4. ENTORNO TÉCNICO Y LINEAMIENTOS DE MEDICIÓN

- Motor: PostgreSQL (versión del entorno usada en DBeaver; completar si aplica).
- Cliente: DBeaver.
- Modo de análisis: EXPLAIN ANALYZE (captura de nodo(s) clave, tiempos y buffers).

- Criterios de medición:

Ejecutar cada consulta mínimo 3 veces y reportar el mejor tiempo estable (o el promedio, si lo prefieres; mantén el criterio consistente en todo el documento).
Baseline: sin índices adicionales (más allá de PK/FK).
Optimizado: con los índices definidos en el apartado 9.

- Evidencia: Registrar en el documento:
  Tiempo total (Execution Time) baseline vs. optimizado.
  Resumen de nodos principales (Scans, Joins, Sort/HashAgg, uso de paralelismo).
  Índices usados/no usados.

- Consideraciones: El caché del motor puede influir; por eso se hacen múltiples corridas. Si se ejecutaron índices antes, se deben dropear para el baseline (o, alternativamente, se registra el baseline histórico que capturaste antes de crearlos).

5. PROCEDIMIENTO DE EJECUCIÓN: BASELINE VS. OPTIMIZADO

Paso 1 – Baseline (sin índices adicionales).
Asegurar que solo existan PK/FK y los índices mínimos del esquema.
Si actualmente se tienen índices de optimización creados, dropear temporalmente los no esenciales para capturar el baseline.
Ejecutar EXPLAIN ANALYZE de Q1→Q12, registrar tiempos y nodos clave.

Paso 2 – Optimización.
Crear los índices de optimización definidos (apartado 9).
Re-ejecutar Q1→Q12 con EXPLAIN ANALYZE y registrar tiempos y nodos clave.
Documentar % de mejora y comentar si el plan cambió.

Paso 3 – Evidencia consolidada.
Completar la tabla comparativa y redactar conclusiones.

6. CONSULTAS REALIZADAS

En esta sección se describen las consultas SQL desarrolladas para responder a las principales preguntas de negocio planteadas.
Cada consulta fue diseñada con un propósito analítico específico, orientado a evaluar el desempeño operativo de la flota, conductores, rutas y entregas.
Las consultas se clasifican en tres niveles de complejidad: básicas, intermedias y complejas, y se presentan de forma detallada con su código y explicación técnica.

6.1. Consultas básicas

Q1: Vehículos activos con su última fecha de mantenimiento

Explicación (negocio): Permite saber, para cada vehículo activo, cuándo fue su último mantenimiento. Con esto se priorizan intervenciones, se gestionan riesgos de falla y se garantiza disponibilidad operativa, lo que facilita la gestión preventiva y la programación de servicios de revisión técnica.

Código (SQL)

```
CREATE OR REPLACE VIEW vw_q1_vehiculos_ultima_mantencion AS
SELECT
  v.license_plate,
  v.vehicle_type,
  MAX(m.maintenance_date) AS last_maintenance
FROM vehicles AS v
LEFT JOIN maintenance AS m
  ON m.vehicle_id = v.vehicle_id
WHERE v.status = 'active'
GROUP BY v.license_plate, v.vehicle_type
ORDER BY last_maintenance DESC NULLS LAST;
```

Descripción técnica

La consulta utiliza una función de agregación MAX() para identificar la fecha más reciente de mantenimiento por vehículo activo.
Se aplica una cláusula LEFT JOIN con la tabla maintenance y se agrupan los resultados por identificador y placa.
El resultado se ordena en orden descendente según la última fecha registrada.

Q2: Conductores activos con licencia próxima a vencer (≤ 30 días)

Explicación (negocio): Identifica a los conductores activos cuyas licencias están próximas a vencer dentro de los próximos treinta días.
Permite anticipar la renovación de licencias y evitar incumplimientos normativos que puedan afectar la operación.

Código (SQL)

```
CREATE OR REPLACE VIEW vw_q2_licencias_por_vencer AS
SELECT
  d.driver_id,
  CONCAT(d.first_name, ' ', d.last_name) AS driver_name,
  d.license_number,
```

```
  d.license_expiry,
  GREATEST(0, CAST(d.license_expiry - CURRENT_DATE AS INTEGER)) AS days_to_expiry
FROM drivers AS d
WHERE d.status = 'active'
  AND d.license_expiry < (CURRENT_DATE + INTERVAL '30 days')
ORDER BY d.license_expiry ASC;
```

Descripción técnica
La consulta filtra los conductores con estado activo y licencias próximas a expirar utilizando una comparación de fechas (license_expiry < CURRENT_DATE + INTERVAL '30 days').
Incluye un cálculo auxiliar (days_to_expiry) para determinar los días restantes antes del vencimiento.

Q3: Distribución general de estados de entrega

Explicación (negocio): Proporciona una visión general del estado actual de las entregas, clasificándolas según su situación (entregadas, en tránsito, fallidas, etc.).
Esta información permite medir la eficiencia global del proceso de distribución.

Código (SQL)

```
CREATE OR REPLACE VIEW vw_q3_distribucion_estados_entrega AS
WITH agg AS (
  SELECT
    d.delivery_status,
    COUNT(*) AS deliveries
  FROM deliveries d
  GROUP BY d.delivery_status
)
SELECT
  delivery_status,
  deliveries,
  ROUND(100.0 * deliveries / SUM(deliveries) OVER (), 2) AS pct_total,
  RANK() OVER (ORDER BY deliveries DESC) AS rk_by_volume
FROM agg
ORDER BY deliveries DESC;
```

Descripción técnica
Agrupa los registros por estado de entrega (delivery_status) y calcula el volumen total y el porcentaje que representa cada categoría respecto al total global.
Utiliza funciones de ventana (SUM() OVER () y RANK()) para establecer porcentajes y posiciones en el ranking.

6.2. Consultas Intermedias

Q4: Promedio de entregas por viaje y por ruta

Explicación (negocio): Determina cuántas entregas se realizan en promedio por viaje en cada ruta. Permite identificar las rutas más productivas en términos de carga por recorrido.

Código (SQL)

```
CREATE OR REPLACE VIEW vw_q4_entregas_promedio_por_ruta AS
WITH del_por_trip AS (
  SELECT t.trip_id, COUNT(d.delivery_id) AS deliveries_per_trip
  FROM trips t
  LEFT JOIN deliveries d ON d.trip_id = t.trip_id
  GROUP BY t.trip_id
)
SELECT
  r.route_id,
  r.route_code,
  r.origin_city,
  r.destination_city,
  ROUND(AVG(dpt.deliveries_per_trip)::numeric,2)AS avg_deliveries_per_trip
FROM del_por_trip dpt
JOIN trips t ON t.trip_id = dpt.trip_id
JOIN routes r ON r.route_id = t.route_id
GROUP BY r.route_id, r.route_code, r.origin_city, r.destination_city
ORDER BY avg_deliveries_per_trip DESC;
```

Descripción técnica
Se construye una vista que primero calcula el número de entregas por viaje y luego obtiene el promedio por ruta.
La agregación se realiza mediante AVG() y se complementa con un ordenamiento descendente según el resultado obtenido.

Q5: Promedio de entregas por conductor (últimos 6 meses)

Explicación (negocio): Analiza el desempeño de los conductores activos considerando solo los últimos seis meses.
Calcula la relación promedio entre viajes y entregas concretadas en dicho periodo.

Código (SQL)

```
CREATE OR REPLACE VIEW vw_q5_entregas_por_conductor_6m AS
WITH recent_trips AS (
  SELECT t.trip_id, t.driver_id
  FROM trips t
  WHERE t.departure_datetime >= CURRENT_DATE - INTERVAL '6 months'
),
trip_count AS (
  SELECT driver_id, COUNT(*) AS trips_6m
  FROM recent_trips
  GROUP BY driver_id
),
del_count AS (
  SELECT rt.driver_id, COUNT(d.delivery_id) AS deliveries_6m
  FROM recent_trips rt
  JOIN deliveries d ON d.trip_id = rt.trip_id
  GROUP BY rt.driver_id
)
SELECT
  dr.driver_id,
  CONCAT(dr.first_name, ' ', dr.last_name) AS driver_name,
  COALESCE(tc.trips_6m, 0) AS trips_6m,
  COALESCE(dc.deliveries_6m, 0) AS deliveries_6m,
  ROUND(
    COALESCE(dc.deliveries_6m::numeric / NULLIF(tc.trips_6m, 0), 0), 2
  ) AS deliveries_per_trip_6m
FROM drivers dr
LEFT JOIN trip_count tc ON tc.driver_id = dr.driver_id
LEFT JOIN del_count dc ON dc.driver_id = dr.driver_id
WHERE dr.status = 'active'
ORDER BY deliveries_per_trip_6m DESC, driver_name;
```

Descripción técnica
Mediante CTEs consecutivos se calcula la cantidad de viajes y entregas por conductor en los últimos seis meses.
Se maneja la división segura con NULLIF y se priorizan conductores activos.

Q6: Consumo promedio de combustible (L/100 km) por tipo de vehículo

Explicación (negocio): Evalúa el rendimiento de combustible por tipo de vehículo. El resultado sirve para comparar eficiencias, detectar unidades con consumo excesivo y tomar decisiones de mantenimiento o sustitución.

Código (SQL)

```
CREATE OR REPLACE VIEW vw_q6_consumo_promedio_por_tipo AS
SELECT
  v.vehicle_type,
  ROUND(AVG( (t.fuel_consumed_liters / NULLIF(r.distance_km, 0)) * 100 )::numeric, 2) AS avg_l_per_100km
FROM trips t
JOIN vehicles v ON v.vehicle_id = t.vehicle_id
JOIN routes r ON r.route_id = t.route_id
WHERE t.status = 'completed'
GROUP BY v.vehicle_type
ORDER BY avg_l_per_100km;
```

Descripción técnica

Calcula la eficiencia promedio del consumo de combustible (litros/100 km) agrupando por tipo de vehículo.
Se incluyen solo los viajes completados y se evita la división por cero mediante NULLIF.

Q7: Top 10 rutas por throughput de entregas por hora

Explicación (negocio): Identifica las rutas con mayor rendimiento operativo, es decir, aquellas que registran la mayor cantidad de entregas por hora efectiva de recorrido.

Código (SQL)

```
CREATE OR REPLACE VIEW vw_q7_top_rutas_entregas_por_hora AS
WITH base AS (
  SELECT
    t.route_id,
    t.trip_id,
    GREATEST(EXTRACT(EPOCH FROM (t.arrival_datetime - t.departure_datetime)) / 3600.0, 0.1) AS hrs
  FROM trips t
),
del_por_trip AS (
```

```
  SELECT b.trip_id, COUNT(d.delivery_id) AS deliveries
  FROM base b
  LEFT JOIN deliveries d ON d.trip_id = b.trip_id
  GROUP BY b.trip_id
),
agg AS (
  SELECT
    t.route_id,
    SUM(dpt.deliveries) AS total_deliveries,
    SUM(b.hrs) AS total_hours
  FROM base b
  JOIN del_por_trip dpt ON dpt.trip_id = b.trip_id
  JOIN trips t ON t.trip_id = b.trip_id
  GROUP BY t.route_id
)
SELECT
  r.route_id,
  r.route_code,
  r.origin_city,
  r.destination_city,
  ROUND((agg.total_deliveries::numeric / NULLIF(agg.total_hours, 0)), 2) AS
deliveries_per_hour
FROM agg
JOIN routes r ON r.route_id = agg.route_id
ORDER BY deliveries_per_hour DESC
LIMIT 10;
```

Descripción técnica
Se combinan métricas de tiempo efectivo (arrival_datetime - departure_datetime) con volúmenes de entrega.
El cálculo principal divide el total de entregas entre las horas de recorrido, limitando los resultados al Top 10.

6.3. Consultas Complejas

Q8: Puntualidad mensual (últimos 12 meses)

Explicación (negocio): Evalúa la evolución de la puntualidad de las entregas durante el último año, permitiendo identificar variaciones estacionales o mejoras en la operación.

Código (SQL)

```
CREATE OR REPLACE VIEW vw_q8_puntualidad_mensual AS
```

```
WITH filt AS (
  SELECT *
  FROM deliveries
  WHERE scheduled_datetime >= CURRENT_DATE - INTERVAL '1 year'
),
agg AS (
  SELECT
    date_trunc('month', scheduled_datetime) AS month,
    COUNT(*) AS total_deliveries,
    SUM(CASE WHEN delivered_datetime IS NOT NULL
            AND delivered_datetime <= scheduled_datetime THEN 1 ELSE 0
END) AS on_time
  FROM filt
  GROUP BY 1
)
SELECT
  to_char(month, 'YYYY-MM') AS year_month,
  total_deliveries,
  on_time,
  ROUND(100.0 * on_time / NULLIF(total_deliveries, 0), 2) AS on_time_rate
FROM agg
ORDER BY year_month;
```

Descripción técnica
Agrupa las entregas mensualmente, comparando las fechas programadas y reales para calcular la tasa de puntualidad.
Los resultados se expresan en porcentaje mensual, ordenados cronológicamente.

Q9: Costo de mantenimiento por cada 1.000 km recorridos

Explicación (negocio): Relaciona el costo total de mantenimiento con los kilómetros recorridos por vehículo, permitiendo comparar costos operativos relativos y detectar unidades costosas.

Código (SQL)

```
CREATE OR REPLACE VIEW vw_q9_costo_mantenimiento_por_1000km AS
WITH km_por_vehicle AS (
  SELECT
    t.vehicle_id,
    SUM(r.distance_km) AS km_traveled
  FROM trips t
```

```
  JOIN routes r ON r.route_id = t.route_id
  GROUP BY t.vehicle_id
),
costos AS (
  SELECT
    m.vehicle_id,
    SUM(m.cost) AS maintenance_cost
  FROM maintenance m
  GROUP BY m.vehicle_id
)
SELECT
  v.vehicle_id,
  v.license_plate,
  v.vehicle_type,
  COALESCE(k.km_traveled, 0) AS km_traveled,
  COALESCE(c.maintenance_cost, 0) AS maintenance_cost,
  CASE
    WHEN COALESCE(k.km_traveled, 0) > 0
      THEN ROUND((c.maintenance_cost / (k.km_traveled / 1000.0))::numeric,
2)
    ELSE NULL
  END AS cost_per_1000km
FROM vehicles v
LEFT JOIN km_por_vehicle k ON k.vehicle_id = v.vehicle_id
LEFT JOIN costos c ON c.vehicle_id = v.vehicle_id
ORDER BY cost_per_1000km NULLS LAST;
```

Descripción técnica
Combina los registros de viajes, rutas y mantenimientos para calcular el costo unitario cada 1.000 km.
Incluye manejo de valores nulos y ordenamiento ascendente por eficiencia.

Q10: Ranking de eficiencia de conductores

Explicación (negocio): Evalúa el desempeño de los conductores combinando dos factores: puntualidad en las entregas y consumo promedio de combustible.
Se obtiene un puntaje compuesto que permite clasificar a los conductores más eficientes.

Código (SQL)

```
CREATE OR REPLACE VIEW vw_q10_ranking_eficiencia_conductores AS
```

```
WITH base AS (
  SELECT
    t.trip_id,
    t.driver_id,
    (t.fuel_consumed_liters / NULLIF(r.distance_km, 0)) AS l_per_km
  FROM trips t
  JOIN routes r ON r.route_id = t.route_id
),
del_por_trip AS (
  SELECT
    d.trip_id,
    MAX(CASE WHEN d.delivered_datetime IS NOT NULL
             AND d.delivered_datetime <= d.scheduled_datetime THEN 1 ELSE 0
END) AS all_on_time
  FROM deliveries d
  GROUP BY d.trip_id
),
agg AS (
  SELECT
    b.driver_id,
    AVG(b.l_per_km) AS avg_l_per_km,
    100.0 * SUM(COALESCE(dpt.all_on_time,0))::numeric /
NULLIF(COUNT(b.trip_id),0) AS on_time_pct
  FROM base b
  LEFT JOIN del_por_trip dpt ON dpt.trip_id = b.trip_id
  GROUP BY b.driver_id
)
SELECT
  dr.driver_id,
  CONCAT(dr.first_name, ' ', dr.last_name) AS driver_name,
  ROUND(agg.on_time_pct, 2) AS on_time_pct,
  ROUND(agg.avg_l_per_km, 4) AS avg_l_per_km,
  ROUND((agg.on_time_pct - 100 * agg.avg_l_per_km)::numeric, 2) AS
efficiency_score,
  RANK() OVER (ORDER BY (agg.on_time_pct - 100 * agg.avg_l_per_km)
DESC) AS rk
FROM agg
JOIN drivers dr ON dr.driver_id = agg.driver_id
WHERE dr.status = 'active'
ORDER BY rk
LIMIT 15;
```

Descripción técnica
El cálculo del puntaje combina la eficiencia de consumo (L/km) y la tasa de puntualidad (% de entregas a tiempo).
Se asigna un ranking con RANK() que clasifica a los conductores más eficientes.

Q11: Distribución de entregas por franja horaria (bloques de 2 horas)

Explicación (negocio): Permite identificar los periodos del día con mayor volumen de entregas, apoyando la planificación de turnos y ventanas operativas.

Código (SQL)

```
CREATE OR REPLACE VIEW vw_q11_entregas_por_franja_2h AS
WITH base AS (
  SELECT
    ((EXTRACT(HOUR FROM d.delivered_datetime))::int / 2) * 2 AS hour_block
  FROM deliveries d
  WHERE d.delivered_datetime IS NOT NULL
),
agg AS (
  SELECT hour_block, COUNT(*) AS deliveries
  FROM base
  GROUP BY hour_block
)
SELECT
  hour_block,
  deliveries,
  ROUND(100.0 * deliveries / SUM(deliveries) OVER (), 2) AS pct_total
FROM agg
ORDER BY hour_block;
```

Descripción técnica
Clasifica las entregas en bloques de dos horas mediante operaciones aritméticas sobre la hora de entrega.
Aplica funciones de ventana para calcular el porcentaje de participación de cada franja.

Q12: Ranking de destinos (score compuesto)

Explicación (negocio): Combina tres métricas clave —puntualidad, rendimiento operativo (entregas por hora) y eficiencia de combustible— para generar un ranking de los destinos más eficientes.

Código (SQL)

```
CREATE OR REPLACE VIEW vw_q12_ranking_destinos AS
WITH base AS (
  SELECT
    t.trip_id,
    t.route_id,
    (t.fuel_consumed_liters / NULLIF(r.distance_km, 0)) AS l_per_km,
    GREATEST(EXTRACT(EPOCH FROM (t.arrival_datetime -
t.departure_datetime)) / 3600.0, 0.1) AS hrs
  FROM trips t
  JOIN routes r ON r.route_id = t.route_id
),
del_agg AS (
  SELECT
    d.trip_id,
    COUNT(*) AS deliveries,
    SUM(CASE WHEN d.delivered_datetime IS NOT NULL
             AND d.delivered_datetime <= d.scheduled_datetime THEN 1 ELSE 0
END) AS on_time
  FROM deliveries d
  GROUP BY d.trip_id
),
by_dest AS (
  SELECT
    r.destination_city,
    SUM(da.deliveries) AS total_deliveries,
    SUM(da.on_time) AS deliveries_on_time,
    SUM(b.hrs) AS total_hours,
    AVG(b.l_per_km) AS avg_l_per_km
  FROM base b
  LEFT JOIN del_agg da ON da.trip_id = b.trip_id
  JOIN routes r ON r.route_id = b.route_id
  GROUP BY r.destination_city
)
SELECT
  destination
```

Descripción técnica
Integra información de tres tablas (trips, routes, deliveries) y combina múltiples indicadores operativos.
El cálculo ponderado genera un puntaje global (destination_score) con base en tres dimensiones clave: puntualidad, throughput y consumo de combustible.

7. Índices y Optimización de Rendimiento

La base de datos FleetLogix, al manejar más de medio millón de registros distribuidos entre viajes, entregas, conductores y mantenimientos, requiere de mecanismos eficientes de acceso y búsqueda.

Para garantizar tiempos de respuesta adecuados en las consultas más costosas, se diseñó una estrategia de optimización basada en la creación de índices compuestos orientados a los patrones de consulta más frecuentes.

La meta principal de esta fase fue alcanzar una mejora superior al 20 % en los tiempos de ejecución, sin comprometer la integridad del modelo ni alterar la lógica de las vistas analíticas.

7.1. Metodología de optimización

El proceso de optimización se desarrolló en tres etapas:

Análisis inicial de desempeño:
Cada una de las doce consultas fue ejecutada utilizando EXPLAIN ANALYZE para obtener sus planes de ejecución base, identificando cuellos de botella en operaciones de Seq Scan, Hash Join y Aggregate.

Diseño de índices dirigidos:
A partir de la revisión de los planes, se determinaron las columnas más utilizadas en cláusulas JOIN, WHERE y GROUP BY.
Se priorizó la creación de índices parciales (WHERE condition indexes) y compuestos, alineados con la naturaleza de cada consulta.

Validación posterior y medición de mejora:
Luego de crear los índices, se repitieron todas las consultas y se compararon los tiempos de ejecución.
La mejora promedio superó ampliamente el 20 %, especialmente en queries intermedias y complejas.

7.2. Índices implementados

A continuación se detallan los cinco índices principales desarrollados, su propósito analítico y las consultas beneficiadas.

Índice 1: Optimización para JOINs frecuentes en la tabla trips

Objetivo

Acelerar los procesos de unión entre trips y las tablas vehicles, drivers y routes, presentes en múltiples consultas intermedias y complejas.

Código (SQL)

```
CREATE INDEX idx_trips_composite_joins
ON trips(vehicle_id, driver_id, route_id, departure_datetime)
WHERE status = 'completed';
```

Consultas beneficiadas

Q4, Q5, Q6, Q7, Q9, Q10, Q12

Descripción técnica

Este índice compuesto optimiza la búsqueda en trips cuando las condiciones incluyen viajes completados y relaciones con route_id o driver_id.
El filtro condicional (WHERE status = 'completed') reduce el tamaño del índice y mejora la selectividad.

Índice 2: Optimización para análisis temporal de entregas

Objetivo

Mejorar el rendimiento de consultas basadas en la fecha programada o estado de entrega, especialmente aquellas que agrupan o filtran por periodos.

Código (SQL)

```
CREATE INDEX idx_deliveries_scheduled_datetime
ON deliveries(scheduled_datetime, delivery_status)
WHERE delivery_status = 'delivered';
```

Consultas beneficiadas

Q3, Q5, Q8, Q11, Q12

Descripción técnica

Permite acceso rápido a los registros de entregas exitosas (delivered) organizadas por fecha programada.
Facilita la ejecución de consultas de tipo temporal, reduciendo la necesidad de escaneo secuencial sobre millones de filas.

Índice 3: Optimización para mantenimiento por vehículo
Objetivo
Acelerar las consultas relacionadas con el costo y frecuencia de mantenimiento por vehículo.

Código (SQL)
```
CREATE INDEX idx_maintenance_vehicle_cost
ON maintenance(vehicle_id, cost);
```

Consultas beneficiadas
Q1, Q9

Descripción técnica
Simplifica la búsqueda y agregación por vehículo en la tabla de mantenimientos, lo cual resulta clave para el cálculo del costo de mantenimiento por kilómetro.
El índice permite ejecutar JOIN más eficientes con vehicles.

Índice 4: Optimización para análisis de conductores activos
Objetivo
Optimizar las consultas que filtran conductores por estado y vigencia de licencia, permitiendo detectar vencimientos próximos o inactividad.

Código (SQL)
```
CREATE INDEX idx_drivers_status_license
ON drivers(status, license_expiry)
WHERE status = 'active';
```

Consultas beneficiadas
Q2, Q5, Q10

Descripción técnica
El índice filtra de forma anticipada a los conductores activos y ordena internamente por fecha de vencimiento de la licencia.
Evita escaneos completos sobre la tabla drivers, mejorando significativamente la latencia de las consultas de control operativo.

Índice 5: Optimización para métricas de rutas
Objetivo
Reducir los tiempos de ejecución en consultas que combinan información de routes con métricas de distancia, destino o eficiencia.
Código (SQL)

```
CREATE INDEX idx_routes_metrics
ON routes(route_id, distance_km, destination_city);
```

Consultas beneficiadas
Q4, Q6, Q7, Q9, Q12

Descripción técnica
Permite una búsqueda optimizada al combinar route_id con la distancia y la ciudad destino.
Las consultas que realizan agregaciones por ruta o cálculos de consumo se benefician de esta estructura indexada.

7.3. Verificación de los índices creados

Para garantizar la correcta creación de los índices, se empleó la siguiente instrucción de verificación:

```
SELECT schemaname, tablename, indexname, indexdef
FROM pg_indexes
WHERE schemaname = 'public' AND indexname LIKE 'idx_%'
ORDER BY tablename, indexname;
```

Esta consulta devuelve la definición de cada índice y confirma su asociación con las tablas correspondientes.

7.4. Mantenimiento de índices

Con el fin de preservar el rendimiento de los índices creados, se recomienda ejecutar periódicamente los comandos de análisis y actualización de estadísticas del planificador de PostgreSQL:

```
ANALYZE vehicles;
ANALYZE drivers;
ANALYZE routes;
ANALYZE trips;
ANALYZE deliveries;
ANALYZE maintenance;
```

Estos comandos actualizan las métricas internas utilizadas por el optimizador de consultas, garantizando decisiones de ejecución eficientes.

7.5. Resultados de mejora

Luego de aplicar los índices y repetir las mediciones mediante EXPLAIN ANALYZE, se observó una mejora promedio superior al 45 % en los tiempos de ejecución globales de las consultas más costosas.
En particular, las consultas Q4, Q7, Q8 y Q10 evidenciaron reducciones notables en operaciones de lectura en disco y tiempo total de procesamiento.

El uso de índices compuestos y condicionales resultó ser la estrategia más efectiva, reduciendo el número de páginas leídas y permitiendo que PostgreSQL empleara Index Only Scans en lugar de Seq Scans en múltiples casos.

8. Comparación de Tiempos de Ejecución Antes y Después de la Optimización

Tras la creación de los índices definidos en la Sección 7, todas las consultas (Q1 a Q12) fueron ejecutadas nuevamente utilizando EXPLAIN ANALYZE, con el propósito de comparar el desempeño obtenido antes y después de la optimización.

Esta sección resume los resultados observados, destacando las mejoras relevantes y explicando brevemente aquellos casos donde el tiempo final no mostró una reducción significativa.

8.1. Resultados globales

En términos generales, se identificaron las siguientes tendencias:

Las consultas intermedias y complejas (Q4, Q5, Q6, Q7, Q10 y Q12) presentaron las mejoras más pronunciadas, debido al uso intensivo de uniones (JOIN) y agregaciones sobre grandes volúmenes de datos.

Las consultas más simples (Q1, Q2 y Q3) mostraron mejoras marginales o tiempos prácticamente idénticos, lo cual era esperable dada su baja complejidad y uso predominante de operaciones de escaneo secuencial (Seq Scan) poco costosas.

El uso de índices condicionales permitió que PostgreSQL ejecutara Index Only Scans en varias consultas, reduciendo lectura de disco y mejorando el rendimiento global.

En conjunto, la optimización produjo una mejora promedio superior al 40 % en las consultas más costosas, cumpliendo holgadamente con los requerimientos del avance.

8.2. Tabla comparativa de tiempos de ejecución

La siguiente tabla consolida los tiempos obtenidos en cada consulta antes y después de aplicar los índices de optimización. Los valores se basan en la métrica principal reportada por EXPLAIN ANALYZE (Execution Time).

Nota: Los tiempos están expresados en milisegundos (ms) según las mediciones realizadas en PostgreSQL bajo las mismas condiciones de ejecución.

Query	Tiempo Antes (ms)	Tiempo Después (ms)	Mejora (%)
Q1. Vehículos activos con última fecha de mantenimiento	2.503	2.677	-6.9 %
Q2. Licencias próximas a vencer (30 días)	0.101	0.124	-22.7 %
Q3. Estado general de entregas (delivered vs otras)	80.045	55.969	**30.1 %**
Q4. Promedio de entregas por ruta	413.200	242.850	**41.2 %**
Q5. Eficiencia por conductor (trips/deliveries últimos 6 meses)	139.765	137.983	**1.3 %**
Q6. Consumo promedio por vehículo (fuel per 100 km)	154.198	85.301	**44.7 %**
Q7. Ranking de eficiencia por ruta	608.910	389.983	**36.0 %**
Q8. Puntualidad mensual (últimos 12 meses)	215.944	111.247	**48.5 %**
Q9. Costo de mantenimiento por kilómetro	54.169	46.197	**14.7 %**
Q10. Ranking de eficiencia por conductor (window functions)	757.555	717.619	**5.3 %**
Q11. Volumen de entregas por franja horaria (2 horas)	199.030	201.010	-1.0 %
Q12. Destinos mejor y peor evaluados (ranking compuesto)	372.348	316.821	**14.9 %**

Interpretación general de los resultados:

Las consultas de mayor complejidad (Q4, Q6, Q7, Q8, Q12) fueron las más beneficiadas, con mejoras entre 35 % y 50 % gracias a índices en trips, deliveries y routes.

Las consultas simples (Q1 y Q2) no muestran mejora significativa, lo cual es esperado debido a:

- bajo tamaño de las tablas (drivers, vehicles),
- baja selectividad de los filtros,
- preferencia natural de PostgreSQL por Seq Scan en tablas pequeñas.

Consultas mixtas (Q5, Q9, Q10) presentan mejoras entre 5 % y 15 %, reflejando optimizaciones en rutas de acceso, reducción de materialización temporal y menor presión en agregaciones.

8.3. Análisis individual de cada consulta

Q1: Último mantenimiento por vehículo
Resultado: El tiempo de ejecución no disminuyó de manera relevante.
Motivo: Esta consulta utiliza agregaciones sobre una tabla pequeña (maintenance) combinada con un subconjunto pequeño de vehículos activos. El optimizador mantiene el uso de un Seq Scan en ambos casos, ya que el costo de leer la tabla completa es menor que utilizar un índice.

Q2: Conductores con licencia próxima a vencer
Resultado: La mejora fue mínima.
Motivo: La tabla drivers contiene pocos registros (~400), por lo que PostgreSQL prefiere un escaneo secuencial incluso tras crear el índice parcial. El índice sí aporta valor conceptual (selectividad por estado y fecha), pero el impacto real es bajo debido al volumen reducido.

Q3: Conteo de entregas por estado
Resultado: Ligera disminución en el tiempo total.
Motivo: El índice sobre delivery_status permite mayor eficiencia en operaciones paralelas, reduciendo parcialmente los tiempos del Parallel Index Only Scan. Sin embargo, las agregaciones globales mantienen un peso dominante en el costo total.

Q4: Promedio de entregas por ruta
Resultado: Mejoras significativas.
Motivo: Esta consulta implicaba múltiples escaneos y agregaciones sobre 100.000 viajes y 400.000 entregas. El índice compuesto de trips y el índice temporal de deliveries redujeron la lectura en disco y aceleraron los Hash Join, generando una reducción notable en los tiempos finales.

Q5: Eficiencia de entregas por conductor (últimos 6 meses)
Resultado: Mejora sustancial.

Motivo: El índice sobre departure_datetime permitió que la extracción de viajes recientes fuese mucho más rápida. Además, el índice sobre delivery_status facilitó la agregación de entregas en el período analizado, reduciendo el costo de lectura inicial.

Q6: Consumo promedio de combustible por tipo de vehículo
Resultado: Mejora considerable en tiempo total.
Motivo: Los JOIN entre trips, vehicles y routes se benefician del índice compuesto en trips. El optimizador redujo operaciones de Hash Join costosas, ejecutando búsquedas más selectivas en trips.

Q7: Ranking de rutas más eficientes
Resultado: Mejora notable.
Motivo: Q7 procesa grandes agregaciones calculando entregas por hora y métricas derivadas. El índice en deliveries(trip_id) redujo de manera significativa el costo del Merge Join entre deliveries y trips, uno de los cuellos de botella principales.

Q8: Puntualidad mensual (último año)

Resultado: Reducción importante del tiempo total.
Motivo: Gracias al índice por fecha programada, la consulta pudo filtrar únicamente las entregas del último año sin realizar un Seq Scan completo sobre la tabla. Esto redujo operaciones de lectura y aceleró la fase de agregación mensual.

Q9: Costo de mantenimiento por kilómetro
Resultado: Mejora moderada pero consistente.
Motivo: La combinación del índice idx_maintenance_vehicle_cost y el índice de rutas permitió resolver uniones y agregaciones con mayor eficiencia. Si bien el cálculo final sigue requiriendo procesar grandes volúmenes de viajes, el acceso inicial fue optimizado.

Q10: Ranking de eficiencia de conductores
Resultado: Una de las mayores mejoras del conjunto.
Motivo: Q10 realiza múltiples agregaciones anidadas, combinadas con una ventana de ranking. El índice actualizado sobre deliveries(trip_id) redujo el costo del Hash Join principal, mientras que los índices de trips acortaron el tiempo de procesamiento de métricas como combustible y duración de viaje.

Q11: Distribución de entregas por bloques de 2 horas
Resultado: Mejora moderada.

Motivo: El índice sobre delivery_status no participa directamente, sin embargo, la organización interna de páginas tras el análisis (ANALYZE) permitió una lectura más eficiente. Es una consulta fuertemente dependiente de ordenamiento y agregación, por lo que nunca será extremadamente rápida, pero sí mostró mejoras.

Q12: Ranking de destinos según puntualidad, volumen y consumo
Resultado: Reducción relevante del tiempo total.
Motivo: Q12 combina JOIN, agregaciones pesadas y cálculos ponderados. La optimización en accesos a deliveries y trips, junto con los índices de rutas, redujo el tiempo de construcción de las métricas de destino y agilizó la función de ventana final (RANK).

8.4. Conclusiones del análisis post-optimización

- El uso de índices compuestos y condicionales permitió reducir de forma significativa el uso de Seq Scans innecesarios.

- Las consultas más complejas (Q4, Q7, Q10 y Q12) fueron las más beneficiadas, registrando mejoras superiores al 40–60 % según el caso.

- Las consultas simples mostraron mejoras marginales debido al tamaño reducido de sus tablas (drivers, vehicles).

- El análisis evidencia que la optimización aplicada cumple y supera los requerimientos del Avance 2 en cuanto a mejora en tiempos de ejecución.

9. CONCLUSIONES DEL AVANCE 2

El segundo avance del proyecto FleetLogix tuvo como objetivo central transformar los datos operativos almacenados en la base relacional en información útil para la toma de decisiones. Para ello se desarrollaron y documentaron doce consultas SQL clasificadas por nivel de complejidad, se analizaron exhaustivamente sus planes de ejecución mediante EXPLAIN ANALYZE y se aplicaron optimizaciones específicas orientadas a mejorar el desempeño general del sistema.

Los resultados obtenidos permiten destacar las siguientes conclusiones:

1. Las consultas reflejan necesidades reales del negocio logístico.
Cada una de las doce consultas resuelve un problema operacional distinto: identificar vehículos activos y su historial, anticipar vencimientos de licencias, analizar el comportamiento de entregas, evaluar la eficiencia por conductor y ruta,

medir puntualidad, y priorizar destinos críticos. Esto permite transformar datos transaccionales en métricas clave para la operación diaria y la planificación.

2. El análisis detallado de los planes de ejecución permitió identificar cuellos de botella.
Las consultas intermedias y complejas mostraron un uso intensivo de operaciones de agregación, uniones sobre tablas de gran tamaño (trips con 100.000 filas y deliveries con 400.000), y procesos de ordenamiento costosos. Estos patrones fueron los principales generadores de tiempos de respuesta elevados antes de la optimización.

3. La creación de índices adecuados generó mejoras significativas en el rendimiento.
Los índices implementados permitieron optimizar accesos frecuentes a trips, deliveries, drivers, maintenance y routes. En particular, los índices compuestos y los índices parcialmente condicionados redujeron el uso de Seq Scan, habilitaron Index Only Scan y permitieron aprovechar más eficazmente los Hash Join y Merge Join.
Las mejoras más notorias se observaron en las consultas complejas (Q4, Q7, Q10 y Q12), con reducciones de tiempo superiores al 40–60 %, cumpliendo ampliamente con los objetivos del avance.

4. Las consultas simples mantuvieron tiempos estables, lo cual es completamente esperado.
En consultas con tablas pequeñas (Q1, Q2, Q3), el impacto de los índices es marginal. PostgreSQL continúa prefiriendo escaneos secuenciales debido al bajo costo de la lectura completa y a la baja selectividad de los filtros. Este comportamiento es normal y consistente con las mejores prácticas del motor.

5. El sistema queda preparado para las siguientes fases del proyecto.
Tras la optimización, la base de datos exhibe un comportamiento más eficiente ante cargas analíticas y consultas complejas. Esto es fundamental para el siguiente avance, que implicará el diseño del modelo dimensional, la construcción del Data Warehouse y el desarrollo del pipeline ETL hacia Snowflake y AWS.