Avance #2

Creación de Querys:

La consigna principal es crear 8 Querys usando el motor de PostgreSQL. Se crearon 12 Querys de las cuales 3 son básicas, 5 son intermedias y 4 son difíciles. Además, luego de hacer la consulta y obtener el resultado se aplicó la función EXPLAIN ANALYZE

Query 1

La primer Query busca contar cuantos vehículos hay por cada tipo de vehículo

```
-- Query 1: Contar vehículos por tipo
-- Resultado esperado: Lista simple con tipos de vehículo y cantidades.
explain analyze
select
    v.vehicle_type as Tipo_Vehiculo,
    count(v.vehicle_id) as Cantidad_Vehiculos
from resources.vehicles v
group by v.vehicle_type;
```

Resultado:

	A-Z tipo_vehiculo	123 cantidad_vehiculos
1	Camión Grande	60
2	Van	69
3	Motocicleta	20
4	Camión Mediano	51

Execution Time Explain Analyze

- Sin index: 0,286 ms
- Con índex: 11,560 ms

Query 2

La segunda Query busca reconocer los conductores que su licencia esta próxima a vencer, esto quiere decir que se vence en los próximos 30 días. Esta consulta puede servir como una view para poder recordarle a los conductores que deben renovar su licencia antes de su fecha de vencimiento

```sql
-- Query 2: Conductores con licencia próxima a vencer
-- Resultado esperado: Lista de conductores que deben renovar licencia en 30 días.
explain analyze
select
    d.driver_id,
    concat(d.first_name, ' ', d.last_name) as Nombre,
    d.license_expiry,
    d.status
from persons.drivers d
where (d.license_expiry between current_timestamp and (current_timestamp + '30 days')) and d.status = 'active'
order by d.license_expiry;
```

Resultado:

	driver_id	nombre	license_expiry	status
1	27	Lorena Díaz	2025-12-17	active
2	65	Camilo Álvarez	2025-12-17	active
3	250	Joan Muñoz	2025-12-21	active
4	239	David Espitia	2025-12-25	active
5	338	Marlon González	2025-12-25	active
6	102	Elizabeth Bermúdez	2025-12-30	active

Execution Time Explain Analyze

- Sin index: 0,257 ms
- Con índex: 0,133 ms

Query 3

La tercer Query busca ver cuáles son las ciudades que más viajes realizan lo que podemos observar que la ciudad con más viajes realizados es Bogotá, pero igual la empresa tiene una media de viajes realizados bastante pareja

```sql
-- Query 3: Total de viajes por estado
-- Resultado esperado: Conteo simple por estado (in_progress, completed)
explain analyze
select
    r.destination_city as Ciudad_Viaje,
    count(t.trip_id) as Viajes,
    t.status as Estado
from resources.trips t
join resources.routes r on r.route_id = t.route_id
group by r.destination_city, t.status;
```

Resultado:

	ciudad_viaje	viajes	estado
1	Cartagena	18.781	completed
2	Cali	18.731	completed
3	Medellín	18.679	completed
4	Bogotá	24.874	completed
5	Barranquilla	18.935	completed

Execution Time Explain Analyze

- Sin index: 40,827 ms
- Con índex: 43,775 ms

Query 4

Esta busca saber el total de entregas en el periodo de los últimos 60 días, una métrica bastante interesante y útil. Nos puede servir para conocer la productividad como un factor de entregas por mes aquí use la función “trim” la cual elimina los espacios en blanco (o caracteres especificados) del principio, del final o de ambos lados de una cadena de texto entonces antes de la coma se eliminan los valores y por eso nos queda solo la ciudad, esto se hizo con el fin de no tener que usar Joins con las demás tablas

```
-- Query 4: Total de entregas por ciudad (últimos 2 meses, 60 días)
-- Resultado esperado: Ranking de ciudades con volumen de entregas y peso total.
explain analyze
select
    count(d.delivery_id) as Viajes,
    trim(split_part(d.delivery_address, ',', -1)) as Ciudad,
    sum(d.package_weight_kg) as Total_KG
from resources.deliveries d
where d.delivered_datetime >= current_timestamp - interval '60 days'
group by ciudad
order by viajes desc;
```

Resultado:

	viajes	ciudad	total_kg
1	1.921	Bogotá	780.045,06
2	1.606	Medellín	681.227,51
3	1.474	Cali	627.853,95
4	1.408	Barranquilla	579.457,86
5	1.388	Cartagena	577.570,79

Execution Time Explain Analyze

- Sin index: 136,532 ms
- Con índex: 145,225 ms

Query 5

Esta Query nos sirve para algo importante y es conocer los conductores activos y la cantidad de viajes que realizan. Conoces también algo importante y es que tenemos 370 Conductores Activos que realicen viajes, solo puse 10 en la imagen ya que no caben los 370 en un screenshot 👌

```sql
-- Query 5: Conductores activos y carga de trabajo
-- Resultado esperado: Lista con total de viajes por conductor activo.
explain analyze
select
    concat(d.first_name, ' ', d.last_name) as Nombre,
    count(t.route_id) as Viajes
from persons.drivers d
join resources.trips t
    on d.driver_id = t.driver_id
where d.status = 'active'
group by Nombre
order by Viajes desc;
```

Resultado:

	A-Z nombre	123 viajes
1	Alejandra Montoya	560
2	José Pérez	542
3	Andrés Álvarez	498
4	Carmen Blanco	311
5	Yaneth Castrillón	311
6	Luz Castro	308
7	Antonio Rentería	302
8	Humberto Cerón	302
9	Gabriela Vargas	302
10	Tania Calderón	302

Execution Time Explain Analyze

- Sin index: 65,450 ms
- Con índex: 51,240 ms

Query 6

A partir de esta Query se ponen nivel avanzado las demás, igual esta lógica la usaremos más adelante como métrica. Para una empresa como fleetlogix es

importante saber cuatas entregas en promedio hace cada conductor para así saber si están siendo eficientes o no y así mismo tomar medidas, ocurre lo mismo de la Query pasada al ser 370 Conductores diferentes no me caben en un screenshoot. Por cierto, utilice un intervalo entre el **current_timestamp** y los próximos **6 months** para que así se actualice cada día la consulta

```
-- Query 6: Promedio de entregas por conductor (6 meses)
-- Resultado esperado: Métricas de productividad por conductor.
explain analyze
select
    concat(t3.first_name, ' ', t3.last_name) as Nombre,
    round(avg(t1.delivery_id), 0) as Promedio_Entregas,
    sum(t2.total_weight_kg) as Total_KG,
    sum(t2.fuel_consumed_liters) as Consumo
from resources.deliveries t1
join resources.trips t2
    on t1.trip_id = t2.trip_id
join persons.drivers t3
    on t2.driver_id = t3.driver_id
where t1.delivered_datetime >= current_timestamp - interval '6 months'
group by nombre
order by promedio_entregas desc;
```

Resultado:

	A-Z nombre	123 promedio_entregas	123 total_kg	123 consumo
1	Omar Martínez	368.743	362.814,65	12.672,64
2	Julio Ortiz	367.978	330.561,96	10.671,87
3	Enrique Salcedo	367.839	388.772,6	15.344,73
4	Francisco Luna	367.727	394.601,77	12.690,13
5	Giovanny Castañeda	367.608	278.087,46	10.733,7
6	Sandra Flórez	367.599	501.790,49	16.697,73
7	Pablo Castillo	367.271	305.670,69	9.537,6
8	Marlon González	367.009	401.279,48	15.480,64
9	Carmen Suárez	366.817	284.692,89	10.879,14
10	Felipe Mercado	366.730	472.768,11	16.184,03

Execution Time Explain Analyze

- Sin index: 165,516 ms
- Con índex: 153,575 ms

Query 7

Otra de las métricas de las que usaremos la lógica más adelante en las Querys Avanzadas o difíciles. Aquí para calcular el consumo en 100Km use una lógica sencilla cogí el total de litros consumidos y los dividí por la cantidad de km recorridos y así tengo la cantidad de litros que se consumen por Km luego simplemente lo multiplique por 100 y listo. Una lógica sencilla y funcional

```sql
-- Query 7: Rutas con mayor consumo de combustible
-- Resultado esperado: Top 10 rutas con mayor consumo litros/100km.
explain analyze
select
    r.route_code,
    r.origin_city,
    r.destination_city,
    round(sum(r.distance_km), 0) as Total_Km,
    round(sum(t.fuel_consumed_liters), 0) as Consumo,
    round((sum(t.fuel_consumed_liters)/sum(r.distance_km))*100, 2) as Consumo_en_100Km
from resources.routes r
join resources.trips t
    on r.route_id = t.route_id
group by r.route_code, r.origin_city, r.destination_city
order by consumo_en_100km desc
limit 10;
```

Resultado:

	route_code	origin_city	destination_city	total_km	consumo	consumo_en_100km
1	R009	Bogotá	Barranquilla	966.004	112.033	11,6
2	R037	Barranquilla	Cali	998.528	115.601	11,58
3	R016	Medellín	Cali	1.007.817	116.660	11,58
4	R011	Bogotá	Cartagena	2.204.926	254.931	11,56
5	R003	Bogotá	Medellín	944.334	109.074	11,55
6	R040	Cartagena	Bogotá	2.323.384	268.288	11,55
7	R045	Cartagena	Cali	2.394.114	276.491	11,55
8	R005	Bogotá	Cali	1.033.009	119.255	11,54
9	R025	Cali	Medellín	938.265	108.303	11,54
10	R035	Barranquilla	Medellín	949.582	109.589	11,54

Execution Time Explain Analyze

- Sin index: 79,506 ms
- Con índex: 66,824 ms

Query 8

En esta Query es una métrica importante ya que si la aplicamos a los drivers podemos darnos cuenta del porcentaje de retrasos de cada conductor por día de la semana es interesante saberlo, así nos podemos dar cuenta de que días es más probable que una entrega se retrase por diferentes factores lo que notamos es que el porcentaje más alto son los miércoles y tiene cierto sentido, normalmente son de los días con más tráfico en las grandes ciudades. Aquí utilizamos CTEs para realizar

la Query y usamos una función increíble que es “::numeric” para convertir un numero de un tipo a otro tipo. Además, la ecuación del promedio que usamos es: $\text{promedio} = \frac{\sum x_i}{n}$ solo la multiplicamos por 100 y obtenemos el porcentaje de retrasos

Resultado:

	A-Z dia_semana	123 total_retraso	123 total_entregado	123 porcentaje_retraso
1	SATURDAY	2.764	57.132	4,84
2	WEDNESDAY	2.949	57.650	5,12
3	SUNDAY	2.788	56.856	4,9
4	TUESDAY	2.838	57.004	4,98
5	MONDAY	2.742	56.730	4,83
6	THURSDAY	2.870	57.506	4,99
7	FRIDAY	2.670	57.046	4,68

```
        end as Retraso
    from resources.deliveries d
    where d.delivery_status = 'delivered'
)
select
    to_char(r.scheduled_datetime, 'DAY') as Dia_Semana,
    sum(r.retraso) as Total_Retraso,
    count(r.delivery_id) as Total_Entregado,
    round((sum(r.retraso)::numeric) / count(r.delivery_id) * 100, 2) as porcentaje_retraso
from retrasos r
group by dia_semana;
```

Execution Time Explain Analyze

- Sin index: 284,902 ms
- Con índex: 213,004 ms

Query 9

Mediante el uso de CTEs se creó otra métrica para poder hacer un buen análisis de costos de mantenimiento. Algo bastante importante en una empresa en crecimiento. Lo que vemos que es algo bastante lógico es que el costo de revisión de motor es mucho más alto ya que normalmente son los más complicados y delicados. Aquí la maquina se demoró maso menos 4 segundos en ejecutar la consulta, ojalá que con el índex se demore menos 🤞

```sql
-- Query 9: Costo de mantenimiento por kilómetro
-- Resultado esperado: Costo por km para cada tipo de vehículo usando CTEs.
explain analyze
with costos_mantenimiento as(
    select
        t3.maintenance_type,
        count(t3.maintenance_id) as Cantidad_Mantenimientos,
        sum(t1.distance_km) as Total_Viajado,
        sum(t1.estimated_duration_hours) as Tiempo_Viajado,
        sum(t1.toll_cost) as Costo_Viaje,
        sum(t3."cost") as Costo_Mantenimiento
    from resources.routes t1
    join resources.trips t2
        on t1.route_id = t2.route_id
    join resources.maintenance t3
        on t2.vehicle_id = t3.vehicle_id
    group by maintenance_type
)
select
    maintenance_type as Tipo_Mantenimiento,
    round(costo_mantenimiento / total_viajado * 100, 2) as Costo_por_KM
from costos_mantenimiento;
```

Resultado:

	A-Z tipo_mantenimiento	123 costo_por_km
1	Alineación y balanceo	31.574,53
2	Cambio de aceite	26.156,93
3	Cambio de llantas	78.690,08
4	Mantenimiento general	60.925,11
5	Revisión de frenos	43.764,41
6	Revisión de motor	87.415,31

Execution Time Explain Analyze

- Sin index: 4520,274 ms
- Con índex: 2651,834 ms

Query 10

Esta es en mi opinión la consulta más larga ya que me piden usar diferentes métricas para medir la eficiencia las que escogí fueron consumo_por_km, entregas_por_semana y tiempo_retraso. Naturalmente mientras consumo_por_km sea más bajo será mucho mejor y mientras entregas_por_semana sea más alto será mucho mejor. Lo que se hizo fue mediante la función Rank determinar esto, en el order by se arregla eso. Luego sumo todas las metricas como para obtener un valor de productividad y lo organizo del mejor (ósea el mayor) al peor y hago un limit 20 para solo obtener los primeros 20 conductores

```sql
-- Query 10: Ranking de conductores por eficiencia
-- Resultado esperado: Top 20 conductores con ranking múltiple usando Window Functions (RANK).
explain analyze
with consumo_por_km as(
    select
        concat(d.first_name, ' ', d.last_name) as Nombre,
        sum(t.fuel_consumed_liters) as Total_Consumido,
        sum(r.distance_km) as Total_Km,
        round(sum(r.distance_km) / sum(t.fuel_consumed_liters), 2) as consumo_Km
    from persons.drivers d
    join resources.trips t
        on d.driver_id = t.driver_id
    join resources.routes r
        on r.route_id = t.route_id
    group by Nombre
    order by consumo_km asc
), entregas_por_semana as(
    select
        Nombre,
        round(avg(total_entregas), 2) as Promedio_Entregas
    from (
        select
            concat(t1.first_name, ' ', t1.last_name) as Nombre,
            count(t3.delivery_id) as Total_entregas,
            date_trunc('week', t3.delivered_datetime) as Semana
        from persons.drivers t1
        join resources.trips t2
            on t1.driver_id = t2.driver_id
        join resources.deliveries t3
            on t2.trip_id = t3.trip_id
        where t3.delivery_status = 'delivered'
        group by Nombre, Semana
    ) as Semanas
    group by Nombre
    order by promedio_entregas desc
), tiempo_retraso as(
    select
        concat(t3.first_name, ' ', t3.last_name) as Nombre,
        avg(t1.delivered_datetime - (t1.scheduled_datetime + interval '2 hours')) as promedio_retraso
    from resources.deliveries t1
    join resources.trips t2
        on t1.trip_id = t2.trip_id
    join persons.drivers t3
        on t2.driver_id = t3.driver_id
    where t1.delivered_datetime is not null and t1.delivered_datetime > t1.scheduled_datetime + interval '2 hours'
    group by nombre
    order by promedio_retraso asc
)
select
    nombre,
    rank() over(order by consumo_km asc) as rank_Consumo,
    rank() over(order by promedio_entregas desc) as rank_Entregas,
    rank() over(order by promedio_retraso asc) as rank_Retraso,
    rank() over(order by consumo_km asc) + rank() over(order by promedio_entregas desc) + rank() over(order by promedio_retraso asc) as ranking_total
from consumo_por_km
join entregas_por_semana using(nombre)
join tiempo_retraso using(nombre)
order by ranking_total asc
limit 20;
```

Resultado:

	nombre	rank_consumo	rank_entregas	rank_retraso	ranking_total
1	Gabriela Vargas	18	14	70	102
2	Sandra Salamanca	39	6	85	130
3	Javier Bustos	28	51	56	135
4	Yaneth Castrillón	46	4	92	142
5	Mauricio Quintero	18	121	10	149
6	Sara Correa	35	111	12	158
7	Lorena Osorio	46	32	80	158
8	Katherine Flórez	138	19	4	161
9	Andrés Álvarez	107	3	52	162
10	Michael Reyes	66	57	40	163
11	Lucía Jiménez	18	89	61	168
12	Carlos Camargo	46	16	110	172
13	Aleida Gómez	107	60	18	185
14	Deisy Guevara	84	43	59	186
15	Eduardo Ortega	35	73	82	190
16	Claudia Suárez	107	69	16	192
17	Alexandra Rodríguez	66	81	66	213
18	Camilo Mejía	8	71	159	238
19	Adriana Salcedo	28	212	1	241
20	Gabriel Camargo	11	42	188	241

Execution Time Explain Analyze

- Sin index: 1895,638 ms
- Con índex: 618,555 ms

Query 11

Hacer un análisis de tendencias mensuales puede hacerse sobre la tabla que quieras realmente, para este caso yo utilice la tabla de **deliveries** ya que creo que es la que más análisis mensuales se le deben hacer. la función “nullif” compara dos valores en caso de ser iguales crea un null como resultado, se usó para cuando se vaya a comparar con meses que no existen en la base de datos. Aquí la función lag se usó para recibir los datos del mes anterior y lead los del mes próximos luego se creó una nueva variable que es un porcentaje de variación para ver que tal cambia en el tiempo.

```sql
-- Query 11: Análisis de tendencia mensual
-- Resultado esperado: Tendencia mensual con comparaciones usando LAG/LEAD.
explain analyze
with retraso as (
    select
        date_trunc('month', d.delivered_datetime) as mes,
    case
        when d.delivered_datetime > d.scheduled_datetime + interval '2 hours' then 1 else 0
    end as retraso
  from resources.deliveries d
  where d.delivery_status = 'delivered' and d.delivered_datetime is not null
), month_metrics as (
    select
        mes,
        count(*) as total_entregas,
        sum(retraso) as total_retrasos,
        round(sum(retraso)::numeric / nullif(count(*), 0) * 100, 2) as Porcentaje_Retrasos
    from retraso
    group by mes
    order by mes
)
select
    to_char(mes, 'YYYY-MM') as mes_name,
    total_entregas,
    total_retrasos,
    porcentaje_retrasos,
    lag(total_retrasos) over (order by mes) as retrasos_mes_anterior,
    lead(total_retrasos) over (order by mes) as retrasos_mes_siguiente,
    case
        when lag(total_retrasos) over (order by mes) is null then null
        else round((total_retrasos - LAG(total_retrasos) over (order by mes))::numeric / nullif(lag(total_retrasos) over (order by mes), 0) * 100, 2)
    end as variacion_porcentajes_retrasos_vs_mes_anterior
from month_metrics
order by mes asc;
```

Resultado:

	mes_name	total_entregas	total_retrasos	porcentaje_retrasos	retrasos_mes_anterior	retrasos_mes_siguiente	variacion_porcentajes_retrasos_vs_mes_anterior
1	2023-11	1.113	65	5,84	[NULL]	893	[NULL]
2	2023-12	17.813	893	5,01	65	875	1.273,85
3	2024-01	17.771	875	4,92	893	781	-2,02
4	2024-02	16.730	781	4,67	875	890	-10,74
5	2024-03	17.829	890	4,99	781	832	13,96
6	2024-04	17.178	832	4,84	890	934	-6,52
7	2024-05	17.822	934	5,24	832	839	12,26
8	2024-06	17.459	839	4,81	934	867	-10,17
9	2024-07	17.848	867	4,86	839	911	3,34
10	2024-08	17.929	911	5,08	867	891	5,07
11	2024-09	17.331	891	5,14	911	918	-2,2
12	2024-10	17.719	918	5,18	891	867	3,03
13	2024-11	17.267	867	5,02	918	848	-5,56
14	2024-12	17.790	848	4,77	867	840	-2,19
15	2025-01	17.867	840	4,7	848	810	-0,94
16	2025-02	16.142	810	5,02	840	859	-3,57
17	2025-03	17.846	859	4,81	810	803	6,05
18	2025-04	17.318	803	4,64	859	884	-6,52
19	2025-05	17.854	884	4,95	803	859	10,09
20	2025-06	17.271	859	4,97	884	897	-2,83
21	2025-07	17.868	897	5,02	859	847	4,42
22	2025-08	17.871	847	4,74	897	809	-5,57
23	2025-09	17.326	809	4,67	847	602	-4,49
24	2025-10	12.962	602	4,64	809	[NULL]	-25,59

Execution Time Explain Analyze

- Sin index: 359,862 ms
- Con índex: 212,949 ms

Creación y Explicación de los Índex:

Índex 1:

Este índice afectara a las listas donde se llame la tabla de trips. Esta es una de las tablas mas llamada en los join porque es la que conecta a casi todas las demás tablas. Además, este índex es parcial ya que solo toma los estados completados, esto nos ahorra una condición en el where

create index idx_trips_composite_joins on resources.trips(vehicle_id, driver_id, route_id, departure_datetime) where status = 'completed';

Índex 2

Este índex hace referencia a los deliveries y esta creado precisamente para ver las fechas programadas de entrega, esto ayuda a buscar mas fácil. Ya que muchas de las métricas calculadas usan fechas de entrega

create index idx_deliveries_scheduled_datetime on resources.deliveries(scheduled_datetime, delivery_status) where delivery_status = 'delivered';

Index 3

Este índex se aprovecha en las consultas donde se llama o se calculan métricas relacionadas con el costo de mantenimiento como por ejemplo la Query 9

create index idx_maintenance_vehicle_cost on resources.maintenance(vehicle_id, cost);

Index 4

Este índex se aprovecha principalmente para saber el estado de los drivers eso nos ahorra usar un where para buscar los que ya estén activos, esto a la hora de ejecución nos va a ahorrar mucho tiempo, también se usa para calcular la métrica de conductores con licencias próximas a vencer

```
create index idx_drivers_status_license on persons.drivers(status, license_expiry) where status = 'active';
```

Índex 5

Como ultimo índex era necesario hacer uno para referencias las rutas sobre todo para calculas las métricas que tengan que ver con distancias y ciudades. Ya que estas no son relacionadas directamente con otras, pero si nos sirve para evitar un tiempo de ejecución excesivo cuando por ejemplo calculamos el costo de mantenimiento por km

```
create index idx_routes_metrics on resources.routes(route_id, distance_km, destination_city);
```