**OPTIMIZACIÓN DE BASE DE DATOS**

Muy bien a continuación vamos a plantear las DMLS que nos enviaron en la prueba técnica, y vamos a conversar sobre buenas prácticas tanto para estructura, índices y las mismas consultas

Igual deseamos destacar que al no tener acceso a la data o la base de datos, estaríamos conversando en un plano estrictamente teórico.

Buenas prácticas aplicada a la estructura de tablas e índices:

1. **Creación de índices simples (fecha\_transaccion)**: esta buena práctica si puede tener un impacto significativo en el comportamiento de la DML, el índice lo tuviéramos apuntado al campo fecha\_transaccion.
2. **Implementar llave primaria a la tabla**: aunque las consultas no hacen referencia, sería igualmente una buena práctica implementarlo
3. **Normalización de las tablas:** esta buena práctica no se si realmente se aplique a los casos de usos, ya que no tenemos accesos a la estructura completa, sin embargo, una tabla con menos campos tendrá un mejor comportamiento en función a las consultas DML que se realicen.
4. **Crear particiones de la tabla en función a rangos temporales**: cuando hablamos de lagos de datos, esta recomendación puede ayudar, básicamente se trata de crear particiones en funciones a filtros generales, por ejemplo los temporales, en recursos 0.1, se puede ver una implementación básica.
5. **Crear índices parciales aplicado a rangos temporales:** es una buena práctica crear un índice basado en un subconjunto de datos, por ejemplo en el primer caso de uso sería interesante implementar un índice parcial basado en el rango temporal que hacen referencia, ahí pudiéramos tener un impacto favorable en comportamiento de la dmls.
6. **Creación de índices simples (estado)**: en teoría con la creación de este índice se pudiera mejorar el comportamiento de la segunda DML.
7. **Crear indice parcial (estado)**: se puede crear los mismos con los estados=Completado
8. **Crear índice compuesto (cliente\_id,monto)**: este índice puede optimizar el comportamiento de la segunda DML.
9. **Crear vistas materializadas:** si los datos no tienen casos de usos de seteos, puede ser útil crear estas vistas, para que ya tener solo esas datos, esto puede significar un impacto positivo en el comportamiento de ambas dmls.

A continuación vamos a presentar las DMLS que nos enviaron en la prueba técnica.

SELECT

COUNT(\*)

FROM

transacciones

WHERE

fecha\_transaccion BETWEEN '2024-01-01' AND '2024-06-30';

SELECT

cliente\_id, SUM(monto)

FROM

transacciones

WHERE

estado = 'COMPLETADO'

GROUP BY cliente\_id;

Buenas prácticas aplicadas a las DMLS.

* **Saber cómo se ejecuta la DML:** Nosotros iniciamos optimizaciones sin saber en realidad como se ejecuta la DML, para tener esa información le recomiendo implementar EXPLAIN, en el recursos 1.1 Pueden observar la implementación de ello.
* **Reemplazar \***: en este caso de uso, si existe un valor Null o tupla corrupta igual la va contar, por lo general se estila verificar el campo que deseamos validar, por lo general puede ser la llave primaria de la tabla, en el recurso 1.2 pueden observar la implementación.
* **Incluir AS**: Aunque esta recomendación es más que probable que no haya mejoras en el performance, si tendremos una mejor presentación y sobre todo una gestión en la lógica de negocio que va solicitar ese recurso, en el recurso 1.3 podrán observar la buena práctica.
* **Colocar en nombre de la tabla al utilizar un campo de la misma:** esta buena práctica igualmente no tendrá un impacto en el comportamiento sin embargo, sí puede mejorar la presentación de la DML y la escalabilidad de la misma. Recurso 1.4.

Recursos

**0.1 Implementación de particionamiento**

CREATE TABLE transacciones (

id\_transaccion SERIAL PRIMARY KEY,

fecha\_transaccion DATE,

-- otras columnas

) PARTITION BY RANGE (fecha\_transaccion);

-- Crear particiones por trimestre, por ejemplo:

CREATE TABLE transacciones\_q1\_2024 PARTITION OF transacciones

FOR VALUES FROM ('2024-01-01') TO ('2024-03-31');

CREATE TABLE transacciones\_q2\_2024 PARTITION OF transacciones

FOR VALUES FROM ('2024-04-01') TO ('2024-06-30');

**1.1 Implementación de Explain.**

EXPLAIN SELECT

COUNT(\*)

FROM

transacciones

WHERE

fecha\_transaccion BETWEEN '2024-01-01' AND '2024-06-30';

**1.2 Cambiar el \***

SELECT

COUNT(transacciones.cod)

FROM

transacciones

WHERE

fecha\_transaccion BETWEEN '2024-01-01' AND '2024-06-30';

**1.3 Implementar AS**

SELECT

COUNT(transacciones.cod) AS total\_transacciones

FROM

transacciones

WHERE

fecha\_transaccion BETWEEN '2024-01-01' AND '2024-06-30';

**1.4 Colocar el nombre de la tabla**

SELECT

COUNT(transacciones.cod) AS total\_transacciones

FROM

transacciones

WHERE

fecha\_transaccion BETWEEN '2024-01-01' AND '2024-06-30';