Práctica 09

Optimización de Consultas

1. Objetivo General

Optimizar consultas para reducir el tiempo de respuesta del SMBD.

2. Objetivos Secundarios

- Conocer y aplicar diferentes métodos para optimizar consultas.
- Identificar posibles puntos de optimización en consultas ya definidas.

3. Introducción

Hasta ahora las consultas construidas durante el *Lenguaje de Manipulación de Datos (DML) Básico* (Práctica 07) y el *Lenguaje de Manipulación de Datos (DML) Avanzado* (Práctica 08) regresaban un resultado en un tiempo corto debido a la poca cantidad de registros que existen en la base de datos.

Sin embargo, las bases de datos pueden almacenar una gran cantidad de registros que continuarán aumentando en el tiempo, lo cual podrá incrementar el tiempo de respuesta del SMBD considerablemente al ejecutar una consulta.

Es por ello que es aconsejable analizar la construcción de las consultas antes de ejecutarlas. Sobre todo aquellas en las que se involucre más de una tabla o se tengan que censar condiciones.

Debido a cómo se construye la sintaxis del código SQL para una consulta, en ésta práctica dividiremos en dos grupos a los puntos donde se puede realizar una optimización:

1. *Condiciones (WHERE)*
2. *Tablas (FROM y JOIN)*

3.1. Optimización sobre Condiciones

Los resultados de una consulta se pueden retardar considerablemente si las condiciones que se definen no llevan un orden que permita al SMBD realizar la menor cantidad de cálculos u operaciones. Para ello es necesario considerar factores probabilísticos.

Es importante saber que los factores probabilísticos están dados por el negocio, es decir, el diseñador de la base de datos así como el usuario que construya las consultas no intervienen en la definición de éstos factores.

Tomemos como ejemplo una consulta a una tabla que contiene los datos de los clientes de la Pizzería que reparte a domicilio. Se quiere una consulta que retorne los registros que cumplan las siguientes características simultáneamente:

1. Que el nombre del cliente empiece con la letra E.
2. Que el apellido paterno sea igual al materno.

Suponiendo que la tabla *Cliente* de la base de datos de la pizzería cuenta con una cantidad considerable de registros, entonces el orden en el que se definen las condiciones afectará de manera directa el tiempo de respuesta del SMBD.

Haciendo un análisis se puede apreciar que para realizar la primera condición, el SMBD deberá de leer la primera letra de cada nombre de cada registro y revisar si ésta es la letra E. En la segunda condición el SMBD deberá de leer por cada registro el apellido paterno y compararlo con el apellido materno.

La Tabla 9.1 muestra los factores probabilísticos de la tabla Cliente. Estos factores están definidos por el negocio como se comentó anteriormente, es decir, ya están dadas al momento de realizar la consulta.

Tabla 9.1 - Factores probabilísticos de la tabla Cliente.

Tabla	***Registros***	***Probabilidad***
Cliente	10000	1/500 de que el apellido paterno sea igual al apellido materno

Como la consulta requiere que ambas condiciones se cumplan, es necesario definir primero la condición que consuma menos tiempo y recursos del SMBD. Esto se puede calcular de dos maneras distintas:

1. ***Primer Arreglo*.** Verificar que el apellido paterno sea igual al materno y después realizar la condición del nombre, implicaría:
    a. **Comprobar si el apellido paterno es igual al apellido materno.** Si suponemos que existen 10,000 registros, de acuerdo con la información de la tabla 9.1, entonces tardaríamos 10,000 unidades de tiempo en leer el apellido paterno. Comparar cada apellido paterno implica recorrer cada registro de la columna apellido materno también. Con la suposición anterior, éste proceso nos tomará 10,000 unidades de tiempo adicionales. Si suponemos que con una probabilidad de 1/500 los dos apellidos son iguales, entonces tendríamos, como resultado parcial de ésta comprobación, 20 registros (10,000/500 = 20).
    b. **Comprobar que el nombre de esos 20 registros cumple la condición de comenzar con la letra "E"**. Esto requerirá de 20 unidades de tiempo más.

    Resultado: 10,000(apellido paterno) + 10,000(apellido materno) + 20 (nombre) = ***20,020*** unidades de tiempo.

    La consulta que implementaría este primer arreglo se presenta en la Figura 9.1.

```
SELECT *
FROM Cliente
WHERE Apellido_paterno = Apellido_materno
AND Nombre LIKE 'E%';
```

Figura 9.1 - Consulta a la tabla Cliente.

2. ***Segundo Arreglo***. Verificar que el nombre empiece con la letra "E" y después realizar la comprobación de los apellidos, implicaría:
    a. **Comprobar que el nombre empieza con la letra "E".** Recorrer todos los registros de la tabla Cliente, continuando con la suposición de 10,000 registros tomará 10,000 unidades de tiempo. La probabilidad de que un nombre comience con la letra "E" supongamos es de 1/27 (esta probabilidad viene de suponer que existen 27 letras en él alfabeto), lo cual nos regresaría un total de 370 registros aproximadamente.
    b. **Comprobar si el apellido paterno es igual al apellido materno**. Este proceso se realizará a los 370 registros que se obtuvieron de la condición del punto anterior. Como ya vimos, se tienen que recorrer los registros dos veces para realizar la comprobación uno a uno, lo que llevaría 740 unidades de tiempo en total.

Resultado: 10,000 + (370 * 2) = 10,000 + 740 = ***10,740*** unidades de tiempo.

La consulta que implementaría este primer arreglo se presenta en la Figura 9.2.

```
SELECT *
FROM Cliente
WHERE Nombre LIKE 'E%'
AND  Apellido_paterno = Apellido_materno;
```

Figura 9.2. Consulta optimizada a la tabla Cliente.

Como resultado de éste análisis se aconseja definir la condición de que el nombre comience con la letra E en primer lugar.

3.2. Optimización sobre Tablas

El número de registros que una tabla contiene también puede afectar el tiempo total que tarda el SMBD en realizar una consulta, sobre todo si se trata de una consulta en la que intervienen dos o más tablas.

Tomando como ejemplo las tablas *Orden*, *Orden_por_Cliente*, y *Cliente* de la base de datos de la pizzería procederemos a discernir el mejor orden de asociación de las tablas dependiendo de cada consulta. Por ejemplo: se requiere obtener los datos de todos los clientes tales que:

1. Su nombre sea Edna o Carlos
2. Que en una sola orden hayan comprado 5 artículos o más.

La Tabla 9.2 muestra los factores probabilísticos de las condiciones requeridas, así como el número total de registros que contiene cada una de las tablas necesarias.

Tabla 9.2 - Factores probabilísticos de las tablas Cliente, Orden y Orden_por_Cliente

Tabla	***Registros***	***Probabilidad***
Cliente	10000	1/80 de que el nombre sea Edna o Carlos
Orden	20000	4/5 de que una orden tenga 5 articulos o más
Orden_por_Cliente	20000	

La primera condición se realiza sobre la tabla *Cliente*, trayendo como resultado solo los registros que cumplan con la condición del nombre, lo cual minimiza el número de registros sobre los cuales realizar la segunda comprobación del total de artículos.

Si observamos la segunda condición, notaremos que el número de artículos por orden es mayor a 5 en varios registros, esto se calculó aprovechando la función de agregación AVG. Por lo tanto es muy probable que esta condición arroje como resultado un gran número de registros sobre los cuales se tendría que realizar la comprobación del nombre del cliente.

Como ésta consulta requiere de la información que se encuentra en las tres tablas mencionadas anteriormente es aconsejable definir primero la tabla junto con sus condiciones, de tal manera que se optimicen tanto el tiempo como los recursos del SMBD.

1. ***Primer arreglo***. Verificar en primer lugar dentro de la tabla *Orden* las órdenes que contengan 5 artículos o más para después asociarlos a los respectivos clientes, implicaría:
    a. **Recorrer la tabla *Orden* para comprobar si el total de artículos es mayor o igual a 5**. Suponiendo que una distribución sobre la columna *Total_Artículos* calcula la probabilidad de 4/5 para que una orden tenga 5 artículos o más y que ésta tabla almacena 20,000 registros. Entonces tendríamos como resultado para esta consulta parcial 16,000 registros aproximadamente y habríamos ocupado 20,000 unidades de tiempo al recorrer la tabla.
    b. **Relacionar los registros que arrojó la consulta parcial del punto anterior con los que se encuentran en la tabla *Orden_por_Cliente*.** El *Id_Orden* de cada registro que se obtuvo en la consulta parcial del punto anterior, se tendrá que buscar ahora en la tabla *Orden_por_Cliente* la cual cuenta con 20,000 registros también. El realizar la asociación entre los 16,000 registros de la primera consulta y los 20,000 de la segunda tabla, resulta en un total 320,000,000 de unidades de tiempo. Considerando que el promedio de compras por cliente es de 5 órdenes

podemos estimar entonces que, obtendremos un total de 3200 registros aproximadamente con un *Id_Cliente* distinto.

c. **Verificar a los clientes cuyo nombre sea Edna o Carlos de la tabla *Cliente*.** Los 3200 *Id_Cliente* distintos que arrojó el JOIN de la consulta parcial del punto anterior, serán comprobados uno a uno para hacer la validación de los nombres, lo que consumirá 3,200 unidades de tiempo para cada nombre (Edna y Carlos), totalizando 6,400 unidades de tiempo. Si la probabilidad de que el nombre de un cliente sea Edna o Carlos es de 1/80, entonces tendremos aproximadamente 40 registros como resultado final.

Resultado: 20,000 + (16,000 * 20,000) + (3,200*2)= 20,000 + 320,000,000 + 6,400 = ***320,026,400*** unidades de tiempo.

La consulta que implementaría este primer arreglo se presenta en la Figura 9.3.

```
SELECT Id_Cliente, Nombre, Apellido_paterno, Apellido_materno
FROM Orden NATURAL JOIN Orden_cliente NATURAL JOIN Cliente
WHERE Numero_articulos >= 5
AND Nombre IN ('Edna', 'Carlos');
```

Figura 9.3 - Consulta entre tablas.

2. ***Segundo arreglo***. Verificar en primer lugar dentro de la tabla *Cliente* los clientes cuyo nombre sea Edna o Carlos para después verificar si las órdenes relacionadas con estos clientes tienen 5 artículos o más.
   a. **Verificar los clientes cuyo nombre se Edna o Carlos de la tabla Cliente.** Supongamos que la tabla Cliente almacena 10,000 registros, por lo tanto, tomaría 10,000 unidades de tiempo verificar todos los nombres de la tabla. Recordemos que la probabilidad de que un nombre sea Carlos o Edna es de 1/80, por lo tanto el resultado parcial nos arrojaría 63 registros.
   b. **Relacionar los *Id_Cliente* que arrojó la consulta parcial del punto anterior con los *Id_Orden* que les corresponden.** Suponiendo que la tabla *Orden_por_Cliente* tienen 20,000 registros, asociarlos con los 63 obtenidos anteriormente tomaría 1,260,000 unidades de tiempo. El promedio de órdenes por cliente es de 5, por lo que esperaríamos que el resultado arrojara 315 registros
   c. **Verificar que las órdenes arrojadas por la consulta anterior cumplen la condición de tener 5 o más artículos.** Para relacionar los 315 registros de la consulta parcial del punto anterior con los 20,000 registros de la tabla *Orden* tomaría 6,300,000 unidades de tiempo. La probabilidad de que una orden tenga 5 artículos o más es de 4/5, por lo tanto el total de registros será de 252 aproximadamente.

   Resultado: 10,000 + (20,000 * 63) + (315 * 20,000) = 10,000 + 1,260,000 + 6,300,000 = ***7,570,000*** unidades de tiempo.

La consulta que implementaría este primer arreglo se presenta en la Figura 9.4.

```
SELECT Id_Cliente, Nombre, Apellido_paterno, Apellido_materno
FROM Cliente NATURAL JOIN Orden_cliente NATURAL JOIN Orden
WHERE Nombre IN ('Edna', 'Carlos')
AND Numero_articulos >= 5;
```

Figura 9.4. Consulta optimizada entre tablas.

De esta manera, se observa que conviene definir en primer lugar a la condición que involucra el nombre del cliente de la tabla *Cliente*, para después realizar sobre los registros que fueron obtenidos como resultado parcial de la primera consulta la segunda condición sobre el total de los artículos.

3.3. Índices

El índice de un texto sirve para encontrar de manera rápida un cierto elemento dentro de la totalidad de un contenido. Los índices en las bases de datos tienen una función similar, es decir, mediante el uso de índices se pueden acelerar los tiempos de respuesta de las consultas. Los índices por lo general se construyen para la o las columnas de una cierta tabla que se consulta con cierta frecuencia.

La sintaxis para la creación de índices se muestra en la Figura 9.5.

```
CREATE INDEX INDX_Nombre_cliente
ON Cliente (Nombre);
```

Figura 9.5. Sintaxis utilizada en la creación de índices.

En esta sintaxis encontramos lo siguiente:

1. CREATE INDEX
   Palabras reservadas para comenzar con la instrucción de creado de índices.
2. INDX_Nombre_Cliente
   Es el nombre con el que se define el índice que se está creando.
3. ON
   Palabra reservada para declarar la tabla sobre la cual será creado el índice.
4. Cliente
   Es el nombre de la tabla objetivo del índice.
5. (Nombre)
   Es el nombre de la columna sobre la cual se hará el indexado.

A manera de ejemplo la figura 9.6 muestra la sintaxis para crear el índice que acelere las consultas sobre la tabla *Direccion_Cliente* para la columna *Colonia* de la base de datos de la Pizzería.

Figura 9.6. Sintaxis de ejemplo de índice para la tabla *Dirección_Cliente*.

3.4. EXPLAIN y Query Plan

EXPLAIN es un comando usado para analizar el plan de ejecución de consultas. Este comando genera un reporte de ruta de cómo el motor de PostgreSQL escanea la sentencia, dando el tiempo estimado de ejecución de una fila sobre la que se está realizando la consulta, así como el tiempo total de todas las filas consultadas.

La sintaxis del comando EXPLAIN se presenta en la Figura 9.7.

Figura 9.7 Comando EXPLAIN.

1. EXPLAIN
   Palabra reservada para iniciar con la instrucción de optimización de consulta.
1. SENTENCIA
   Ésta puede ser cualquier consulta hacia la base de datos.
2. ;
   Símbolo ; (punto y coma) indica el final de la instrucción EXPLAIN.

Como ejemplo del comando EXPLAIN, ver Figura 9.8, utilizando tres tablas ya existentes, “Cliente” con un total de 100,000 registros, “Cliente_Producto” con un total de 20,000 registros y “Producto” con 50,000 registros.

```
EXPLAIN SELECT idCliente, nombre, edad, nombreproducto
FROM "Cliente" NATURAL JOIN "Cliente_Producto" NATURAL JOIN "Producto"
WHERE edad > 25 AND nombre IN ('Zia', 'Susan');
```

Figura 9.8 Ejecución del comando EXPLAIN

Para este ejemplo se puede observar plan de ejecución de la consulta (Query Plan), en el que se muestra el tiempo de ejecución (costo), tanto por fila como total, que tomó al SMBD filtrar la información de la condición de las columnas “edad” y “nombre”, así como el requerido para realizar los JOINs con las tres tablas, ver Figura 9.9.

	QUERY PLAN text
1	Nested Loop (cost=0.87..1490.72 rows=15 width=34)
2	Join Filter: ("Cliente".id = "Producto".id)
3	-> Merge Join (cost=0.58..1476.11 rows=30 width=22)
4	Merge Cond: ("Cliente".id = "Cliente_Producto".id)
5	-> Index Scan using "Cliente_pkey" on "Cliente" (cost=0.29..4007.29 rows=148 width=14)
6	Filter: ((edad > 25) AND ((nombre)::text = ANY ('{Zia,Susan}'::text[])))
7	-> Index Scan using "Cliente_Producto_pkey" on "Cliente_Producto" (cost=0.29..640.29 rows=20000 width=8)
8	-> Index Scan using "Producto_pkey" on "Producto" (cost=0.29..0.47 rows=1 width=24)
9	Index Cond: (id = "Cliente_Producto".id)

Figura 9.9 Salida del comando EXPLAIN.

La información arrojada por EXPLAIN resulta de mucha utilidad para el administrador o usuario de la base de datos, ya que el conocer el plan de ejecución de la consulta es posible identificar aquellos puntos más “costosos” y buscar alternativas para reducirlos, todo esto derivando en una optimización de la consulta.

3.5. Conclusión

La optimización de consultas puede reducir el tiempo total de una consulta, así como el consumo de los recursos del SMBD, si se toman en cuenta los siguientes puntos:

1. Evaluar aquellas condiciones cuya probabilidad de ser verdaderas sea menor.
2. Asociar primero las tablas que contengan un menor número de registros o, dependiendo de la naturaleza de la consulta, que arrojen un menor número de registros como resultado.
3. En consultas que tomen un tiempo considerable en ejecutarse, será de gran ayuda indexar aquellas columnas que sean utilizadas como criterio de discriminación en la cláusula WHERE.

4. Ejercicios

(NOTA: Resuelve los siguientes ejercicios en relación al proyecto que realizarás durante el curso, en dado caso que no tengas un proyecto, utiliza la información en el apéndice NFL-ONEFA parte 09 al final de esta práctica para realizarlos)

1. Optimiza cinco consultas de *Lenguaje de Manipulación de Datos (DML) Avanzado* (Práctica 08) en donde se involucren al menos dos condiciones. Justifica tu decisión en términos de operaciones realizadas por registro y unidades de tiempo. Finalmente compara el desempeño de la consulta optimizada contra la consulta original.
2. Optimiza cinco consultas de *Lenguaje de Manipulación de Datos (DML) Avanzado* (Práctica 08) en las que intervengan dos o más tablas. Justifica tu decisión en términos de operaciones realizadas por tupla y unidades de tiempo. Finalmente compara el desempeño de la consulta optimizada contra la consulta original.
3. Crea un **índice para alguna de las tablas de tu base de datos**.

Entregables requeridos para prácticas subsecuentes:

- Índice para alguna de las tablas de tu base de datos

5. Apéndice NFL-ONEFA parte 09

A continuación se presentan 5 consultas que se derivan del esquema NFL-ONEFA. Realiza un análisis y optimización a cada una de ellas y registra la diferencia en los tiempos de ejecución.

```
SELECT nombre_jugador, numero, tds, nombre_equipo
FROM jugador NATURAL JOIN quarterback NATURAL JOIN Equipo NATURAL JOIN Jugador_Equipo
WHERE tds > 5;
```

```
SELECT nombre_ciudad, estado_ciudad
FROM ciudad NATURAL JOIN Equipo_Ciudad NATURAL JOIN Equipo NATURAL JOIN Partido
WHERE Marcador_v - Marcador_L >= 40;
```

```
SELECT nombre_entrenador, marcador_v, marcador_l
FROM Partido NATURAL JOIN Equipo_Entrenador NATURAL JOIN Equipo NATURAL JOIN Entrenador
WHERE marcador_V > 15
OR marcador_l > 15;
```

```
SELECT DISTINCT nombre_ciudad
FROM Equipo_Ciudad NATURAL JOIN Equipo NATURAL JOIN Equipo_Estadio NATURAL JOIN Estadio NATURAL JOIN Ciudad
WHERE Capacidad < 20000
AND nombre_equipo LIKE 'U%';
```

```
SELECT *
FROM jugador NATURAL JOIN Jugador_Equipo NATURAL JOIN Quarterback
WHERE tds > 5
AND apellido_jugador LIKE 'S%'
AND juegos_Jugados < 30;
```

```
SELECT nombre_jugador, apellido_jugador, numero, tackles, ints
FROM jugador NATURAL JOIN defensiva NATURAL JOIN jugador_equipo
WHERE tackles >= 65
OR ints >= 9;
```