FCyT
Sede Concepción del Uruguay

Optimización de Consultas

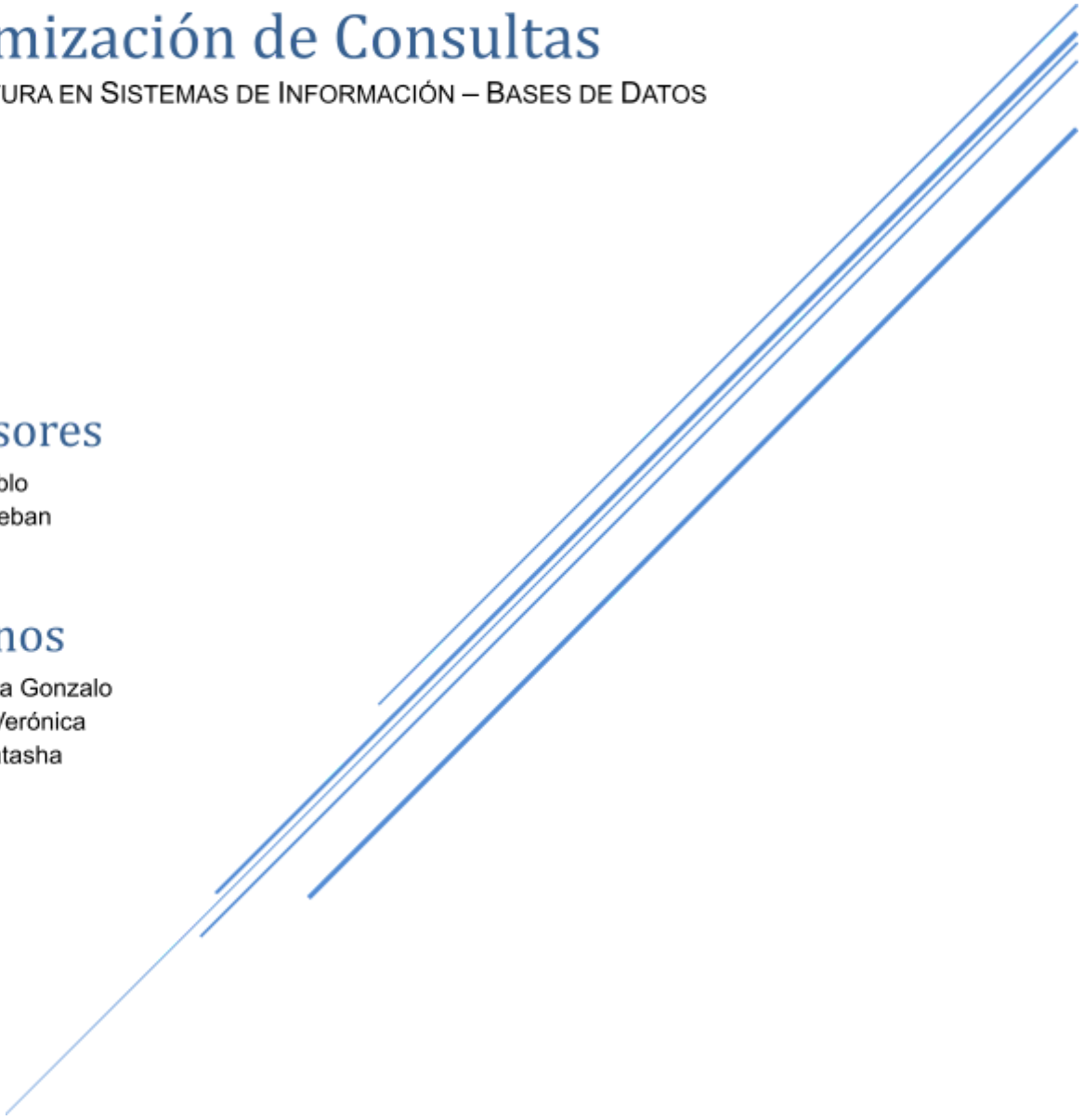
LICENCIATURA EN SISTEMAS DE INFORMACIÓN – BASES DE DATOS

Profesores

Pescio Pablo
Schab Esteban

Alumnos

Errandonea Gonzalo
Gimenez Verónica
Grasso Natasha



2022

a) **Para qué sirve el optimizador de consultas.**
Sirve para mejorar los tiempos de respuesta en un sistema de gestión de bases de datos relacional, ya que la optimización es el proceso de modificar un sistema para mejorar su eficiencia o también el uso de los recursos disponibles.

b) **Qué es un plan de ejecución.**
Un plan de ejecución de consulta es una definición de los siguientes elementos: **La secuencia en la que se tiene acceso a las tablas de origen. Los métodos que se usan para extraer los datos de cada tabla**. Por lo general, hay métodos diferentes para tener acceso a los datos de cada tabla.

c) **Qué tareas realiza el DBMS para analizar una consulta.**
Pasos para el procesamiento de una consulta:

Análisis léxico

- Identificar componentes en el texto de la consulta (SQL)

Análisis sintáctico

- Revisar la sintaxis (corrección gramatical).

Validación semántica

- Verificar validez de nombres de tablas, vistas, atributos.

Traducción de la consulta a una representación interna

- Esto permite que la máquina la manipule mejor, transformar SQL en Álgebra relacional (es lo que generalmente realiza el SGBD en esta etapa).

d) **Qué datos tiene las estadísticas de las tablas de una base de datos.**
Información estadística:

Para cada tabla

- Cardinalidad (n° de tuplas).
- Factor de bloqueo (tuplas que caben en un bloque).
- Bloques ocupados.
- Método de acceso primario y otros métodos de acceso (hash, árbol b, etc.)
- Atributos indexados, de dispersión, de ordenamiento

Para cada atributo

- Valores de dominios distintos almacenados.
- Valores, máximo, mínimo, etc.

e) **Nombre algunas reglas heurísticas para mejorar una consulta.**
- En la optimización heurística se aplican reglas de transformación para modificar la representación interna de la consulta, a fin de mejor el rendimiento.

- Una consulta en Álgebra Relacional se puede expresar de diferentes maneras, obteniéndose similares resultados
- Así también, un lenguaje de consulta comercial, como SQL, permite realizar la misma consulta de diferentes maneras, pero el rendimiento de la ejecución de la consulta, no debe depender de como sea expresada la misma

f) Enumere las buenas prácticas para realizar consultas.

- Ejecutar operaciones de selección σ tan pronto como sea posible.
- Realizar primero las selecciones σ que sean más restrictivas, que generen menor número de columnas.
- Ejecutar las operaciones de proyección Π tan pronto como sea posible.
- Combinar productos cartesianos con una selección, cuya condición representa una condición de reunión, convirtiéndolos en un Join.

g) Busque en la documentación de postgresql los distintos tipos de índices que se pueden generar y en caso utilizaría cada uno de ellos.

Existen varios tipos de índices: B-tree, Hash, GiST, GIN, BRIN. Cada tipo de índice utiliza un algoritmo diferente que se adapta mejor a diferentes tipos de consultas. De forma predeterminada, el comando CREATE INDEX crea índices de B-tree (Árbol B), que se ajustan a las situaciones más comunes.

h) Tomando consultas del ejercicio de elecciones o de libro, el que el grupo prefiera, realice una estimación de costos de consultas más complejas, usando el operador explain en pgadmin para analizar el costo de la consulta, evalúe en grupo si es necesario generar un índice, realícelo y vuelva a ejecutar el explain para ver si mejora la consulta.

Tomando como ejemplo la siguiente consulta de *libros*:

```
select u.nombre, p.fechaPrestamo  from usuario u, localidad l, prestamo p
where l.localidad = 'CONCEPCION DEL URUGUAY'
      and u.idLocalidad= l.idLocalidad
      and p.dni = u.dni
      and p.fechaprestamo < '1991-12-31'
      and p.fechaprestamo > '1988-01-01'
order by u.nombre
```

Obtuvimos los siguientes resultados:

#	Node	Timings		Rows			Loops
		Exclusive	Inclusive	Rows X	Actual	Plan	
1.	→ Sort (cost=27.73..27.73 rows=1 width=26) (actual=0.153..0.1…	0.026 ms	0.154 ms	↓ 15	15	1	1
2.	→ Hash Inner Join (cost=22.96..27.72 rows=1 width=26) (a… **Hash Cond**: (p.dni = u.dni)	0.016 ms	0.128 ms	↓ 15	15	1	1
3.	→ Seq Scan on prestamo as p (cost=0..4.55 rows=54 … **Filter**: ((fechaprestamo < '1991-12-31'::date) AND (fechapr **Rows Removed by Filter**: 116	0.031 ms	0.031 ms	↑ 1	54	54	1
4.	→ Hash (cost=22.94..22.94 rows=1 width=26) (actual=… **Buckets**: 1024 **Batches**: 1 **Memory Usage**: 11 kB	0.01 ms	0.081 ms	↓ 42	42	1	1
5.	→ Hash Inner Join (cost=19.05..22.94 rows=1 wid… **Hash Cond**: (u.idlocalidad = l.idlocalidad)	0.03 ms	0.072 ms	↓ 42	42	1	1
6.	→ Seq Scan on usuario as u (cost=0..3.5 row…	0.021 ms	0.021 ms	↑ 1	150	150	1
7.	→ Hash (cost=19..19 rows=4 width=2) (actu… **Buckets**: 1024 **Batches**: 1 **Memory Usage**: 9 kB	0.009 ms	0.021 ms	↑ 4	1	4	1
8.	→ Seq Scan on localidad as l (cost=0..1… **Filter**: ((localidad)::text = 'CONCEPCION DE **Rows Removed by Filter**: 42	0.013 ms	0.013 ms	↑ 4	1	4	1

Statistics per Node Type

Node type	Count	Time spent	% of query
Hash	2	0.019 ms	12.34%
Hash Inner Join	2	0.046 ms	29.88%
Seq Scan	3	0.065 ms	42.21%
Sort	1	0.026 ms	16.89%

Y luego de crear el siguiente índice:

```
create index on prestamo (fechaprestamo)
```

Obtuvimos los siguientes resultados:

#	Node	Timings		Rows			Loops
		Exclusive	Inclusive	Rows X	Actual	Plan	
1.	→ Sort (cost=27.73..27.73 rows=1 width=26) (actual=0.096..0.0…	0.021 ms	0.098 ms	↓ 15	15	1	1
2.	→ Hash Inner Join (cost=22.96..27.72 rows=1 width=26) (a… **Hash Cond**: (p.dni = u.dni)	0.011 ms	0.078 ms	↓ 15	15	1	1
3.	→ Seq Scan on prestamo as p (cost=0..4.55 rows=54 … **Filter**: ((fechaprestamo < '1991-12-31'::date) AND (fechapr **Rows Removed by Filter**: 116	0.021 ms	0.021 ms	↑ 1	54	54	1
4.	→ Hash (cost=22.94..22.94 rows=1 width=26) (actual=… **Buckets**: 1024 **Batches**: 1 **Memory Usage**: 11 kB	0.005 ms	0.046 ms	↓ 42	42	1	1
5.	→ Hash Inner Join (cost=19.05..22.94 rows=1 wid… **Hash Cond**: (u.idlocalidad = l.idlocalidad)	0.019 ms	0.041 ms	↓ 42	42	1	1
6.	→ Seq Scan on usuario as u (cost=0..3.5 row…	0.013 ms	0.013 ms	↑ 1	150	150	1
7.	→ Hash (cost=19..19 rows=4 width=2) (actu… **Buckets**: 1024 **Batches**: 1 **Memory Usage**: 9 kB	0.002 ms	0.01 ms	↑ 4	1	4	1
8.	→ Seq Scan on localidad as l (cost=0..1… **Filter**: ((localidad)::text = 'CONCEPCION DE **Rows Removed by Filter**: 42	0.008 ms	0.008 ms	↑ 4	1	4	1

Statistics per Node Type

Node type	Count	Time spent	% of query
Hash	2	0.007 ms	7.15%
Hash Inner Join	2	0.03 ms	30.62%
Seq Scan	3	0.042 ms	42.86%
Sort	1	0.021 ms	21.43%

Donde podemos observar que, en efecto, la consulta que utilizó un nuevo índice creado fue más rápida que la anterior.