Tema

Practica Laboratorio

Docente: Marco Antonio Camacho Alatrista

Curso: Base de Datos II

Integrantes:

Gabriel Frank Krisna Zela Flores

Semestre: VI

Ing. de Software

-- EJERCICIO 1

SELECT * FROM pg_indexes WHERE tablename = 'estudiantes_heap';

```
43  -- Verificar que NO tiene índices
44  SELECT * FROM pg_indexes WHERE tablename = 'estudiantes_heap';
45
46  -- 1.4 MEDIR RENDIMIENTO DE BÚSQUEDA SECUENCIAL
47
48  -- Paso 1: Activamos la medición de tiempo para ver cuánto tarda la consulta.
49  \timing on
50
51  -- Paso 2: Realizamos una búsqueda en la tabla heap.
52  SELECT * FROM estudiantes_heap WHERE id_estudiante = 1077;
53
```

Data Output | Messages | Notifications

schemaname name	tablename name	indexname name	tablespace name	indexdef text

SELECT * FROM estudiantes_heap WHERE id_estudiante = 1077;

```
51  -- Paso 2: Realizamos una búsqueda en la tabla heap.
52  SELECT * FROM estudiantes_heap WHERE id_estudiante = 1077;
53
54  -- Paso 3: Ver cuántos registros se examinaron.
55  EXPLAIN (ANALYZE, BUFFERS)
56  SELECT * FROM estudiantes_heap WHERE id_estudiante = 1077;
57
58
```

Data Output | Messages | Notifications

id_estudiante integer	nombres character varying (58)	apellidos character varying (50)	carrera character varying (38)	semestre integer	promedio numeric (4,2)

SELECT * FROM estudiantes_ordenados WHERE id_estudiante BETWEEN 1030 AND 1080;

```
77  SELECT * FROM estudiantes_ordenados WHERE id_estudiante BETWEEN 1030 AND 1080;
78
79  --EJERCICIO 2
80
81
```

Data Output | Messages | Notifications

Showing rows: 1 to 3

	id_estudiante integer	nombres character varying (58)	apellidos character varying (50)	carrera character varying (38)	semestre integer	promedio numeric (4,2)
1	1035	José	Ramirez Cruz	Ingeniería de Software	4	14.90
2	1042	Lucía	Herrera Diaz	Ingeniería Industrial	8	18.10
3	1063	Patricia	Morales Soto	Ingeniería de Sistemas	3	15.40

-- EJERCICIO 2: HASH MANUAL

SELECT id_estudiante,nombres,apellidos, hash_estudiante(id_estudiante) AS posicion_hash

FROM estudiantes

ORDER BY hash_estudiante(id_estudiante), id_estudiante;

```
SELECT id_estudiante,nombres,apellidos, hash_estudiante(id_estudiante) AS posicion_hash
FROM estudiantes
ORDER BY hash_estudiante(id_estudiante), id_estudiante;

-- 2.3 ANALIZAR COLISIONES
SELECT
    hash_estudiante(id_estudiante) AS posicion_hash,
    COUNT(*) AS cantidad_registros,
```

Data Output | Messages | Notifications

Showing rows: 1 to 10

	id_estudiante [PK] integer	nombres character varying (58)	apellidos character varying (50)	posicion_hash integer
1	1001	Ana	Garcia López	0
2	1828	Maria	Torres Vega	1
3	1856	Diego	Castillo Ruiz	1
4	1884	Carmen	Vargas Leon	1
5	1815	Carlos	Mendoza Silva	2
6	1977	Roberto	Jiménez Paz	3
7	1035	José	Ramirez Cruz	6
8	1042	Lucía	Herrera Diaz	6
9	1063	Patricia	Morales Soto	6
10	1098	Miguel	Santos Ríos	6

SELECT

hash_estudiante(id_estudiante) AS posicion_hash, COUNT(*) AS cantidad_registros,

ARRAY_AGG(id_estudiante ORDER BY id_estudiante) AS ids_en_posicion

FROM estudiantes

GROUP BY hash_estudiante(id_estudiante) ORDER BY posicion_hash;

```
-- 2.3 ANALIZAR COLISIONES
SELECT
    hash_estudiante(id_estudiante) AS posicion_hash,
    COUNT(*) AS cantidad_registros,
    ARRAY_AGG(id_estudiante ORDER BY id_estudiante) AS ids_en_posicion
FROM estudiantes
GROUP BY hash_estudiante(id_estudiante)
ORDER BY posicion_hash;
```

Data Output | Messages | Notifications

Showing rows: 1 to 5

	posicion_hash integer	cantidad_registros bigint	ids_en_posicion integer[]
1	0	1	{1001}
2	1	3	{1828,1856,1884}
3	2	1	{1815}
4	3	1	{1977}
5	6	4	{1035,1042,1063,1098}

-- 2.5 INSERTAR DATOS Y VERIFICAR DISTRIBUCIÓN

```
55
56  -- 2.5 INSERTAR DATOS Y VERIFICAR DISTRIBUCIÓN
57
58  INSERT INTO estudiantes_hash
59  SELECT * FROM estudiantes;
```

Data Output | Messages | Notifications

Showing rows: 1 to 5

	schemaname name	tablename name	registros_insertados bigint
1	public	estudiantes_hash	0
2	public	estudiantes_hash_p0	4
3	public	estudiantes_hash_p1	6
4	public	estudiantes_hash_p2	0
5	public	estudiantes_hash_p3	0

-- 3.1 FUNCIÓN PARA INSERTAR DATOS MASIVOS

-- 3.2 COMPARAR RENDIMIENTO DE BÚSQUEDAS

-- HEAP

EXPLAIN (ANALYZE, BUFFERS)

SELECT COUNT(*) FROM estudiantes_heap WHERE id_estudiante = 2500;

```
57  EXPLAIN (ANALYZE, BUFFERS)
58  SELECT COUNT(*) FROM estudiantes_heap WHERE id_estudiante = 2500;
59
60  -- ORDENADO
61  EXPLAIN (ANALYZE, BUFFERS)
62  SELECT COUNT(*) FROM estudiantes_ordenados WHERE id_estudiante = 2500;
```

Data Output | Messages | Notifications

Showing rows: 1 to 8 Page No: 1

	QUERY PLAN text
1	Aggregate (cost=93.19..93.20 rows=1 width=8) (actual time=0.163..0.163 rows=1 loops=1)
2	Buffers: shared hit=43
3	-> Seq Scan on estudiantes_heap (cost=0.00..93.18 rows=4 width=0) (actual time=0.032..0.160 rows=4 loop...
4	Filter: (id_estudiante = 2500)
5	Rows Removed by Filter: 4010
6	Buffers: shared hit=43
7	Planning Time: 0.053 ms
8	Execution Time: 0.177 ms

-- ORDENADO

EXPLAIN (ANALYZE, BUFFERS)

SELECT COUNT(*) FROM estudiantes_ordenados WHERE id_estudiante = 2500;

```sql
-- ORDENADO
EXPLAIN (ANALYZE, BUFFERS)
SELECT COUNT(*) FROM estudiantes_ordenados WHERE id_estudiante = 2500;

-- HASH
EXPLAIN (ANALYZE, BUFFERS)
SELECT COUNT(*) FROM estudiantes_hash WHERE id_estudiante = 2500;
```

Data Output | Messages | Notifications

Showing rows: 1 to 8 | Page No: 1 of 1

	QUERY PLAN text
1	Aggregate (cost=12.10..12.11 rows=1 width=8) (actual time=0.198..0.198 rows=1 loops=1)
2	Buffers: shared hit=4
3	-> Index Only Scan using idx_estudiantes_ordenados_id on estudiantes_ordenados (cost=0.28..12.09 rows=4 width=0) (actual time=0.188..0.193 rows=4 loop…
4	Index Cond: (id_estudiante = 2500)
5	Heap Fetches: 1
6	Buffers: shared hit=4
7	Planning Time: 0.094 ms
8	Execution Time: 0.219 ms

-- HASH

EXPLAIN (ANALYZE, BUFFERS)

SELECT COUNT(*) FROM estudiantes_hash WHERE id_estudiante = 2500;

```sql
-- HASH
EXPLAIN (ANALYZE, BUFFERS)
SELECT COUNT(*) FROM estudiantes_hash WHERE id_estudiante = 2500;
```

Data Output | Messages | Notifications

Showing rows: 1 to 10 | Page No: 1 of 1

	QUERY PLAN text
1	Aggregate (cost=21.71..21.72 rows=1 width=8) (actual time=0.091..0.092 rows=1 loops=1)
2	Buffers: shared hit=10
3	-> Seq Scan on estudiantes_hash_p0 estudiantes_hash (cost=0.00..21.70 rows=4 width=0) (actual time=0.036..0.088 rows=4 loop…
4	Filter: (id_estudiante = 2500)
5	Rows Removed by Filter: 932
6	Buffers: shared hit=10
7	Planning:
8	Buffers: shared hit=3
9	Planning Time: 0.127 ms
10	Execution Time: 0.108 ms

-- 3.3 COMPARAR ESPACIO UTILIZADO

```sql
-- 3.3 COMPARAR ESPACIO UTILIZADO

SELECT
    schemaname,
    relname AS tablename,
    pg_size_pretty(pg_total_relation_size(schemaname || '.' || relname)) AS tamaño
FROM pg_stat_user_tables
WHERE relname LIKE 'estudiantes_%'
ORDER BY pg_total_relation_size(schemaname || '.' || relname) DESC;
```

Data Output | Messages | Notifications

Showing rows: 1 to 7 | Page No: 1 of 1

	schemaname name	tablename name	tamaño text
1	public	estudiantes_ordenados	464 kB
2	public	estudiantes_heap	376 kB
3	public	estudiantes_hash_p3	120 kB
4	public	estudiantes_hash_p2	112 kB
5	public	estudiantes_hash_p1	112 kB
6	public	estudiantes_hash_p0	104 kB
7	public	estudiantes_hash	0 bytes

PREGUNTAS DE ANÁLISIS

1. ¿Qué estructura es más eficiente para búsquedas exactas por ID? ¿Por qué?

La más rápida es la ordenada con índice. Cuando buscamos un ID, el índice permite que vaya directo al registro, sin necesidad de revisar fila por fila. En la tabla Heap toca recorrer toda la tabla aunque en Hash también es veloz, al final el índice sigue siendo la opción más precisa y eficiente.

2. ¿Cuál sería la mejor estructura para consultas que buscan rangos de IDs? Justifique su respuesta.

Para rangos, la mejor es la ordenada porque al estar los datos organizados, Postgre puede leerlos de forma continua y rápida. En Heap tendría que escanear todos los registros uno por uno, y en Hash no sirve mucho porque los datos están repartidos sin un orden definido.

3. Si el sistema requiere insertar 1000 registros por minuto, ¿qué estructura recomendaría? ¿Por qué?

Heap porque esta estructura no tiene que actualizar índices ni mantener orden, así que las inserciones se hacen de manera rápida. En la ordenada se perdería tiempo actualizando el índice, y en Hash también hay un pequeño costo por calcular dónde va cada registro.

4. Analice las colisiones en la tabla hash. ¿Cómo afectan al rendimiento? ¿Qué estrategias propondría para reducirlas?

Las colisiones aparecen cuando varios registros caen en la misma posición del hash. Eso obliga a revisar más datos y puede volver las búsquedas más lentas.
Para reducir este problema se puede usar más particiones para repartir mejor los datos, diseñar una función hash que distribuya de manera más equilibrada.

5. Compare el espacio utilizado por cada estructura. ¿Hay diferencias significativas? ¿A qué se deben?

Sí hay diferencia, Heap ocupa menos porque solo guarda los datos. La ordenada necesita más espacio ya que almacena los datos y también el índice y Hash particionada suele ocupar aún un poco más, porque además de los datos guarda la información de las particiones, esto se debe principalmente a los índices y a cómo PostgreSQL organiza los registros en cada estructura.