Tema

Practica Laboratorio



Docente: Marco Antonio Camacho Alatrista

Curso: Base de Datos II

Integrantes:

Gabriel Frank Krisna Zela Flores

Semestre: VI

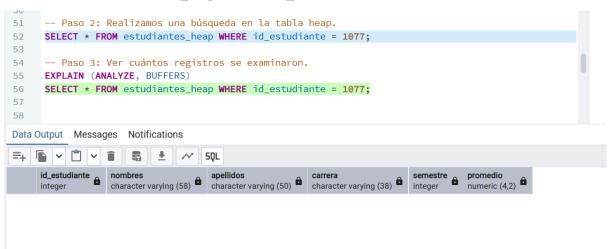
Ing. de Software

-- EJERCICIO 1

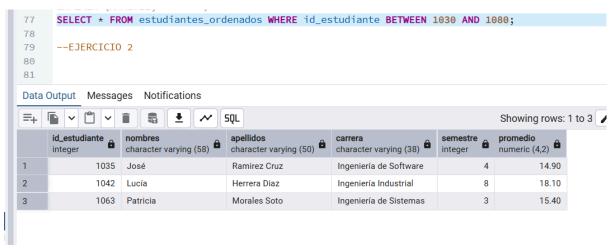
SELECT * FROM pg indexes WHERE tablename = 'estudiantes heap';

```
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.
    \timing on
49
50
51
     -- Paso 2: Realizamos una búsqueda en la tabla heap.
    SELECT * FROM estudiantes_heap WHERE id_estudiante = 1077;
52
53
Data Output Messages Notifications
=+ 🖺 ∨ 📋 ∨ 🛊 👼 👲 ~/ SQL
                                      tablespace
                                                indexdef
                tablename indexname
```

SELECT * FROM estudiantes heap WHERE id estudiante = 1077;



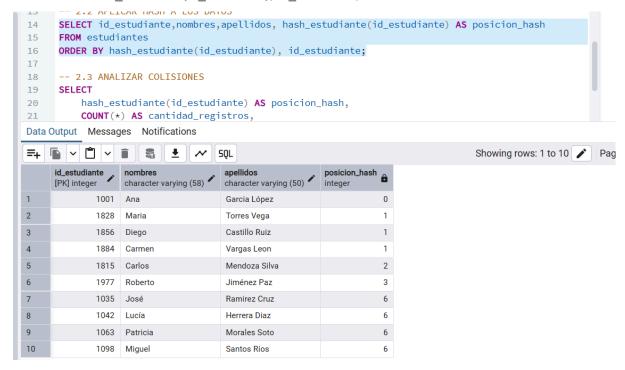
SELECT * FROM estudiantes_ordenados WHERE id_estudiante BETWEEN 1030 AND 1080;



-- 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;



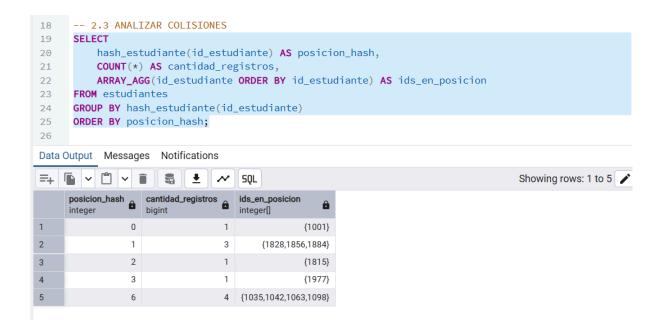
SELECT

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;



-- 2.5 INSERTAR DATOS Y VERIFICAR DISTRIBUCIÓN

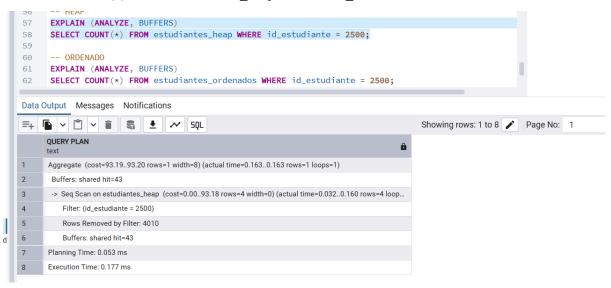


-- 3.1 FUNCIÓN PARA INSERTAR DATOS MASIVOS

- -- 3.2 COMPARAR RENDIMIENTO DE BÚSQUEDAS
- -- HEAP

EXPLAIN (ANALYZE, BUFFERS)

SELECT COUNT(*) FROM estudiantes heap WHERE id estudiante = 2500;



-- ORDENADO

EXPLAIN (ANALYZE, BUFFERS)

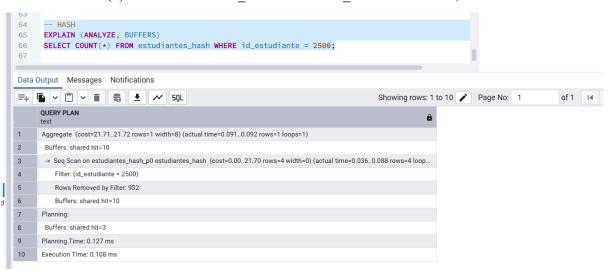
SELECT COUNT(*) FROM estudiantes ordenados WHERE id estudiante = 2500;



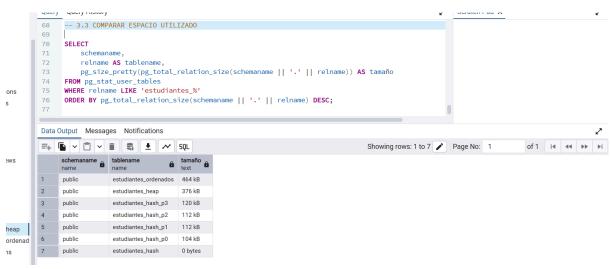
-- HASH

EXPLAIN (ANALYZE, BUFFERS)

SELECT COUNT(*) FROM estudiantes hash WHERE id estudiante = 2500;



-- 3.3 COMPARAR ESPACIO UTILIZADO



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.