



Lab 4

Integrantes:

Jean Piero Huaylla Huillca
Daniel Ignacio Casquino Paz
Jesús Valentín Niño Castañeda

Paso previo: tiempos de creación de índices

Índice B+Tree:

Query utilizada: CREATE INDEX
idx_fecha_btree ON temperatura_btree(fecha);

Tiempo de ejecución: 459ms

```
CREATE INDEX
```

```
Query returned successfully in 459 msec.
```

Índice BRIN:

Query utilizada: CREATE INDEX
idx_fecha_brin ON temperatura_brin USING
BRIN(fecha);

Tiempo de ejecución: 109ms

```
CREATE INDEX
```

```
Query returned successfully in 109 msec.
```

1. (5 pts) Comparar tiempos de ejecución

Rango	B+Tree (ms)	BRIN (ms)
1 minuto	0.076	4.749
1 hora	0.109	3.265
1 dia	0.549	3.839
1 mes	16.188	21.655
2 meses	19.924	29.873

```

6  ▾ EXPLAIN ANALYZE
7  SELECT * FROM temperatura_brin
8  WHERE fecha BETWEEN '2024-06-01 00:00:00' AND '2024-06-01 01:00:00';

```


Data Output Messages Notifications










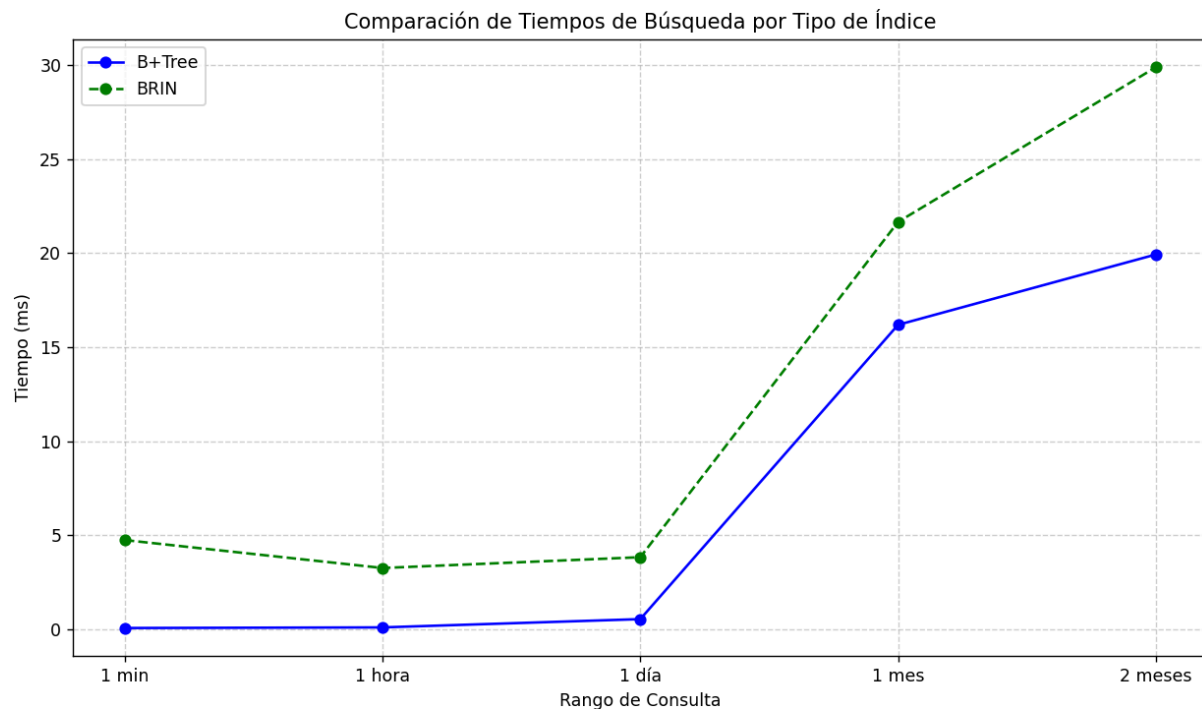

Showing rows: 1 to 8  Page No: 1

	QUERY PLAN	
	text	
1	Bitmap Heap Scan on temperatura_brin (cost=12.05..3652.05 rows=61 width=22) (actual time=3.563..3.884 rows=61 loops=1)	
2	Recheck Cond: (((fecha >= '2024-06-01 00:00:00'::timestamp without time zone) AND (fecha <= '2024-06-01 01:00:00'::timestamp without time ...	
3	Rows Removed by Index Recheck: 20035	
4	Heap Blocks: lossy=128	
5	-> Bitmap Index Scan on idx_fecha_brin (cost=0.00..12.03 rows=19467 width=0) (actual time=0.052..0.053 rows=1280 loops=1)	
6	Index Cond: (((fecha >= '2024-06-01 00:00:00'::timestamp without time zone) AND (fecha <= '2024-06-01 01:00:00'::timestamp without time ...	
7	Planning Time: 0.230 ms	
8	Execution Time: 3.935 ms	

```

05S02. BRIN >  grafica.py > ...
1  import matplotlib.pyplot as plt
2
3  # Datos
4  rangos = ["1 min", "1 hora", "1 día", "1 mes", "2 meses"]
5  tiempos_btree = [0.076, 0.109, 0.549, 16.188, 19.924]
6  tiempos_brin = [4.749, 3.265, 3.839, 21.655, 29.873]
7
8  # Crear la gráfica
9  plt.figure(figsize=(10, 6))
10 plt.plot(rangos, tiempos_btree, marker='o', linestyle='-', color='blue', label='B+Tree')
11 plt.plot(rangos, tiempos_brin, marker='o', linestyle='--', color='green', label='BRIN')
12
13 # Personalización
14 plt.title("Comparación de Tiempos de Búsqueda por Tipo de Índice")
15 plt.xlabel("Rango de Consulta")
16 plt.ylabel("Tiempo (ms)")
17 plt.grid(True, linestyle='--', alpha=0.6)
18 plt.legend()
19 plt.tight_layout()
20
21 # Mostrar la gráfica
22 plt.show()
23

```



2. (1 pts) Medición de tamaño de índices

	Tamaño 2024	Tamaño 2024 y 2025
B+Tree	11 MB	23MB
BRIN	24 kB	24kB

Query utilizada: `SELECT pg_size_pretty(pg_relation_size('idx_fecha_btree')) AS btree, pg_size_pretty(pg_relation_size('idx_fecha_brin')) AS brin;`

	btree text	brin text
1	11 MB	24 kB
1	23 MB	24 kB

Análisis de resultados:

B+Tree

- **Crecimiento según la cantidad de filas:** Al indexar los datos de 2025 junto con los de 2024, el índice B+Tree aumentó el doble, lo que es esperado. Cada fila o grupo de valores necesita espacio en la estructura del árbol. Esto indica que el número de filas se duplicó (probablemente más de 500,000 por año), y el índice también lo hizo.

BRIN

- **Tamaño constante:** Aunque se duplicó las filas, el índice no creció en absoluto. Esto es característico de los BRIN: solo agregan entradas por cada rango de bloques, no por fila. Si las nuevas filas de 2025 están contiguas en disco (por ejemplo, insertadas cronológicamente), simplemente se ajusta el resumen del mismo rango o se añade 1 nuevo, lo cual apenas aumenta el tamaño.

3. (5 pts) Optimización del índice BRIN

Rango	B+Tree (ms)	BRIN-64 (ms)	BRIN-32 (ms)	BRIN-16 (ms)	BRIN-08 (ms)	BRIN-04 (ms)
Space (MB)	23MB	24kB	24kB	32kB	40kB	72kB
1 minuto	0.035ms	1.476ms	1.319ms	0.524ms	0.480ms	0,641ms
1 hora	0.043ms	1.916ms	1.113ms	0.817ms	0.578ms	0.618ms
1 día	0.597ms	2.617ms	0.709ms	0.691ms	0.809ms	0.923ms
1 mes	13.367ms	13.309ms	11.679ms	9.406ms	9.398ms	10.225ms
2 meses	22.584ms	23.296ms	17.235ms	16.377ms	13.224ms	13.589ms

El tiempo de ejecución para la misma consulta y misma pages_per_range fluctuó bastante, por lo que el resultado final es un promedio de 5 ejecuciones. Vemos que el tiempo para rangos pequeños disminuye proporcionalmente al pages_per_range, pero el resultado sigue siendo algo inconsistente.

4. (1 pts) Consulta con filtro compuesto

```
CREATE INDEX idx_sensorid_hash ON temperatura_btree USING HASH (sensor_id);
```

```
SELECT * FROM temperatura_btree
WHERE fecha BETWEEN '2024-06-01' AND '2024-06-02'
AND sensor_id = 5;
```

Si solo existe un índice en sensor_id, el optimizador podría no utilizarlo eficientemente porque también hay un filtro por fecha.

Es probable que aparezca un Seq Scan (secuencial), lo que indica que no se usó el índice.

5. (6 pts) Forzar desorden en datos BRIN

Primer testeo:

```
-- B+ Tree
EXPLAIN ANALYZE
SELECT *
FROM temperatura_btree_desordenada
WHERE fecha BETWEEN '2024-06-01 00:00:00' AND '2024-06-01 01:00:00';
```

B+ Tree: 0.240 ms

```
-- BRIN
EXPLAIN ANALYZE
SELECT *
FROM temperatura_brin_desordenada
WHERE fecha BETWEEN '2024-06-01 00:00:00' AND '2024-06-01 01:00:00';
```

BRIN: 60.987 ms

Tabla:

Rango	B+Tree (ms)	BRIN-64 (ms)	BRIN-32 (ms)	BRIN-16 (ms)	BRIN-08 (ms)	BRIN-04 (ms)
1 minuto	0.043	52.487	46.121	44.047	41.487	54.675
1 hora	0.480	43.165	46.767	44.892	51.459	55.109
1 día	2.290	43.144	42.809	48.115	42.318	51.134
1 mes	17.682	47.315	55.705	47.302	46.435	46.273
2 meses	33.103	49.426	55.113	65.659	80.110	55.396

Para el caso de datos desordenados, en rangos pequeños el índice B+ Tree es muchísimo más rápido que el índice BRIN, sin embargo va aumentando rápidamente conforme se aumenta el tamaño del rango, mientras que el BRIN mantiene su tiempo y hasta parece reducirse ligeramente.

6. (2 pts) Responder las siguientes preguntas

¿En qué escenarios gana el índice BRIN? ¿Por qué?

- Tablas **muy grandes** (millones de filas).
- Columnas con valores **ordenados o correlacionados físicamente** (por ejemplo, una columna fecha donde se insertan cronológicamente)
- Consultas por rangos amplios de tiempo (por ejemplo: fecha BETWEEN '2024-01-01' AND '2024-12-31').

Según la documentación oficial y *PostgreSQL Internals*, BRIN almacena **resúmenes estadísticos por bloques de disco**, no referencias a filas individuales. Esto permite un índice de **tamaño extremadamente reducido**, y buena performance **cuando los datos están físicamente ordenados**.

¿Qué desventajas tiene BRIN frente a B+ Tree?

BRIN solo sabe los rangos mínimos y máximos por bloque, por lo que puede acceder a bloques innecesarios si el resumen es amplio.

- Rendimiento bajo en búsquedas precisas o de pocos valores.
- Dependiente del orden físico de los datos. Si los datos están desordenados, BRIN pierde efectividad.

Como señala PostgreSQL Internals, BRIN puede tener falsos positivos, donde el índice indica que un bloque podría contener datos, aunque realmente no los tenga.

¿Qué pasaría si se mezclan los datos en la tabla con BRIN?

El rendimiento de BRIN caería notablemente.

- Al perderse la correlación física, los resúmenes por bloque ya no reflejarían fielmente los valores reales.
- Esto llevaría a que se lean más bloques innecesarios, disminuyendo la eficiencia del índice.

El manual oficial advierte explícitamente:

“BRIN indexes are most effective on columns whose values are naturally correlated with their physical location within the table.”

Los índices BRIN son más eficaces en columnas cuyos valores están correlacionados de forma natural con su ubicación física dentro de la tabla.

¿Es recomendable usar ambos índices al mismo tiempo?

En general, no es recomendable

- B+Tree y BRIN cumplen funciones distintas y mantener ambos incrementa:
 - Coste de almacenamiento.
 - Coste de mantenimiento (INSERT, UPDATE, VACUUM).
- PostgreSQL solo usa un índice por consulta. Tener ambos puede confundir al planner si no hay una gran diferencia en los costos estimados.

El uso simultáneo podría justificarse si:

- Hay consultas por rango grande (que usan BRIN).

- Y otras por búsquedas exactas o pequeños rangos (que usan B+Tree).
- Y los beneficios superan el coste de mantenimiento.

¿En qué otros casos reales conviene usar BRIN? Fundamente.

Casos reales típicos:

Sistemas de monitoreo o sensores:

- Datos insertados secuencialmente (ej. fecha, temperatura, presión).
- Consultas históricas por periodo.
- BRIN se adapta bien por su mínima huella y eficiencia en series temporales.

Registros de logs (auditorías, transacciones):

- Columnas como timestamp, log_level, etc. suelen estar ordenadas.
- BRIN permite búsquedas por rangos de tiempo con bajo coste.

Almacenamiento de datos científicos o IoT:

- Datos de alta frecuencia donde el volumen crece rápidamente.
- No es viable usar B+Tree por su tamaño.

PostgreSQL Internals señala que BRIN es una alternativa viable a B+Tree para “datasets masivos donde el tamaño de los índices importa más que la velocidad de búsqueda precisa”.