

Base de Datos II Laboratorio 8

Juan Diego Castro, Juan Diego Laredo

28/5/2023

1 Introducción

En gestores de base de datos, como PostgreSQL, es bastante común la utilización de índices para la optimización de consultas. La motivación de este laboratorio es aprender a utilizar el GIN (**Generalized Inverted Index**) para aplicar consultas textuales sobre documentos con la finalidad de recuperar un ranking de documentos basado en el contenido de los documentos y de la query.

2 Sequential Scan vs GIN

2.1 Población de tablas

Nombre de la tabla	Cantidad de tuplas	Índices
lab8_2.articles_1000	1000	gin(body_indexed gin_trgm_ops)
lab8_2.articles_10000	10000	gin(body_indexed gin_trgm_ops)
lab8_2.articles_100000	100000	gin(body_indexed gin_trgm_ops)
lab8_2.articles_1000000	1000000	gin(body_indexed gin_trgm_ops)
lab8_2.articles_10000000	10000000	gin(body_indexed gin_trgm_ops)
lab8_2.articles_100000000	100000000	gin(body_indexed gin_trgm_ops)

```
1 -- populate table with random data
2 INSERT INTO articles_1000
3 SELECT md5(random()::TEXT), md5(random()::TEXT) FROM (
4     SELECT * FROM generate_series(1, 1000) AS id
5 ) AS aux;
```

Llenamos la tabla **articles_1000** con textos generados de forma aleatoria. De la misma forma se procede a llenar las demás tablas.

Tiempos de población de tablas:

```

cs2042db2.lab8_2> INSERT INTO articles_1000
                    SELECT md5(random()::TEXT), md5(random()::TEXT)
                    FROM (SELECT * FROM generate_series(1, 1000) AS id) AS x
[2023-05-28 12:06:36] 1,000 rows affected in 25 ms
cs2042db2.lab8_2> INSERT INTO articles_10000
                    SELECT md5(random()::text), md5(random()::text)
                    FROM (SELECT * FROM generate_series(1, 10000) AS id) AS x
[2023-05-28 12:06:46] 10,000 rows affected in 37 ms
cs2042db2.lab8_2> INSERT INTO articles_100000
                    SELECT md5(random()::text), md5(random()::text)
                    FROM (SELECT * FROM generate_series(1, 100000) AS id) AS x
[2023-05-28 12:06:51] 100,000 rows affected in 169 ms
cs2042db2.lab8_2> INSERT INTO articles_1000000
                    SELECT md5(random()::text), md5(random()::text)
                    FROM (SELECT * FROM generate_series(1, 1000000) AS id) AS x
[2023-05-28 12:06:57] 1,000,000 rows affected in 1 s 728 ms
cs2042db2.lab8_2> INSERT INTO articles_10000000
                    SELECT md5(random()::text), md5(random()::text)
                    FROM (SELECT * FROM generate_series(1, 10000000) AS id) AS x
[2023-05-28 12:07:18] 10,000,000 rows affected in 16 s 315 ms
cs2042db2.lab8_2> INSERT INTO articles_100000000
                    SELECT md5(random()::text), md5(random()::text) FROM (
                        SELECT * FROM generate_series(1, 100000000) AS id
                    ) AS x
[2023-05-28 12:09:52] 100,000,000 rows affected in 2 m 30 s 932 ms

```

Figure 1: Tiempos de población de tablas

Podemos apreciar que, evidentemente, la tabla de 100 millones de registros es la que tarda mas tiempo en poblarse, puesto que la cantidad de textos que debe generar es mucho mayor a las demás.

2.2 Creacion de Índices

En esta parte se procedera a crear un índice en cada una de las tablas. Además, se medirá el tiempo de creación para cada uno de los datasets.

```

1 -- add an index
2 -- Generally Trigram Index
3 CREATE INDEX articles_1000_gin ON articles_1000 USING gin(
4     body_indexed gin_trgm_ops
5 );

```

Nombre de la tabla	Tiempo de creación del índices
lab8_2.articles_1000	24 ms
lab8_2.articles_10000	101 ms
lab8_2.articles_100000	612 ms
lab8_2.articles_1000000	6 s 42 ms
lab8_2.articles_10000000	2 m 8 s 13 ms
lab8_2.articles_100000000	23 m 52 s 134 ms

Table 1: Tiempos de creación de índices

```

cs2042db2.lab8_2> CREATE INDEX articles_1000_gin ON articles_1000 USING gin(body_indexed gin_trgm_ops)
[2023-05-28 12:17:48] completed in 24 ms
cs2042db2.lab8_2> CREATE INDEX articles_10000_gin ON articles_10000 USING gin(body_indexed gin_trgm_ops)
[2023-05-28 12:17:56] completed in 101 ms
cs2042db2.lab8_2> CREATE INDEX articles_100000_gin ON articles_100000 USING gin(body_indexed gin_trgm_ops)
[2023-05-28 12:18:00] completed in 612 ms
cs2042db2.lab8_2> CREATE INDEX articles_1000000_gin ON articles_1000000 USING gin(body_indexed gin_trgm_ops)
[2023-05-28 12:18:09] completed in 6 s 42 ms
cs2042db2.lab8_2> CREATE INDEX articles_10000000_gin ON articles_10000000 USING gin(body_indexed gin_trgm_ops)
[2023-05-28 12:20:25] completed in 2 m 8 s 13 ms
cs2042db2.lab8_2> CREATE INDEX articles_100000000_gin ON articles_100000000 USING gin(body_indexed gin_trgm_ops)
[2023-05-28 12:44:29] completed in 23 m 52 s 134 ms

```

Figure 2: Tiempos de creación de índices

2.3 Tiempos de ejecución

Para medir los tiempos de ejecución, se ha propuesto la siguiente consulta:

- los artículos cuyo **body** contenga el número **3004** (30/04)

```

1 SELECT count(*) FROM articles_100000000 WHERE body ILIKE '%3004%';
2 SELECT count(*) FROM articles_100000000 WHERE body_indexed ILIKE '%3004%';

```

	QUERY PLAN
1	Finalize Aggregate (cost=1756411.97..1756411.98 rows=1 width=8) (actual time=28666.230..28669.471 rows=1 loops=1)
2	-> Gather (cost=1756411.75..1756411.96 rows=2 width=8) (actual time=28666.131..28669.461 rows=3 loops=1)
3	Workers Planned: 2
4	Workers Launched: 2
5	-> Partial Aggregate (cost=1755411.75..1755411.76 rows=1 width=8) (actual time=28662.662..28662.664 rows=1 loops=3)
6	-> Parallel Seq Scan on articles_100000000 (cost=0.00..1755401.33 rows=4167 width=0) (actual time=1.774..28660.869 rows=14740 loops=3)
7	Filter: (body ~* '%3004% '::text)
8	Rows Removed by Filter: 33318593
9	Planning Time: 0.154 ms
10	Execution Time: 28669.501 ms

Figure 3: Planificación de consulta sin índice para 100 millones de registros

	QUERY PLAN
1	Aggregate (cost=38699.68..38699.69 rows=1 width=8) (actual time=324.055..324.058 rows=1 loops=1)
2	-> Bitmap Heap Scan on articles_100000000 (cost=1393.50..38674.68 rows=10000 width=0) (actual time=102.991..320.834 rows=44424 loops=1)
3	Recheck Cond: (body_indexed ~* '%3004% '::text)
4	Rows Removed by Index Recheck: 6607
5	Heap Blocks: exact=50027
6	-> Bitmap Index Scan on articles_100000000_gin (cost=0.00..1391.00 rows=10000 width=0) (actual time=89.095..89.095 rows=51031 loops=1)
7	Index Cond: (body_indexed ~* '%3004% '::text)
8	Planning Time: 0.275 ms
9	Execution Time: 324.106 ms

Figure 4: Planificación de consulta con índice para 100 millones de registros

Lo que podemos concluir de la planificación para 100 millones de registros es que, sin el **GIN**, el proceso es paralelizado haciendo uso de 2 trabajadores que realizan un **Parallel Sequential Scan** en la tabla y van filtrando aquellas tuplas que cumplen la condición (**body ILIKE %3004%**).

Sin embargo, con el índice **GIN**, realiza un **Bitmap Index Scan**, haciendo uso de **articles_100000000_gin**, que es el índice mismo. Además, se puede apreciar claramente la mejora en el tiempo de ejecución con dicho índice.

Nombre de la tabla	Tiempo de ejecución con índice	Tiempo de ejecución sin índice
lab8_2.articles_1000	0.061 ms	0.928 ms
lab8_2.articles_10000	0.076 ms	8.553 ms
lab8_2.articles_100000	0.652 ms	86.430 ms
lab8_2.articles_1000000	4.339 ms	292.712 ms
lab8_2.articles_10000000	31.590 ms	2860.810 ms
lab8_2.articles_100000000	315.496 ms	28439.333 ms

Table 2: Tiempo de ejecución de la consulta en todos los datasets

Como podemos apreciar en la tabla, el comportamiento de los tiempos se mantiene para todos los datasets.

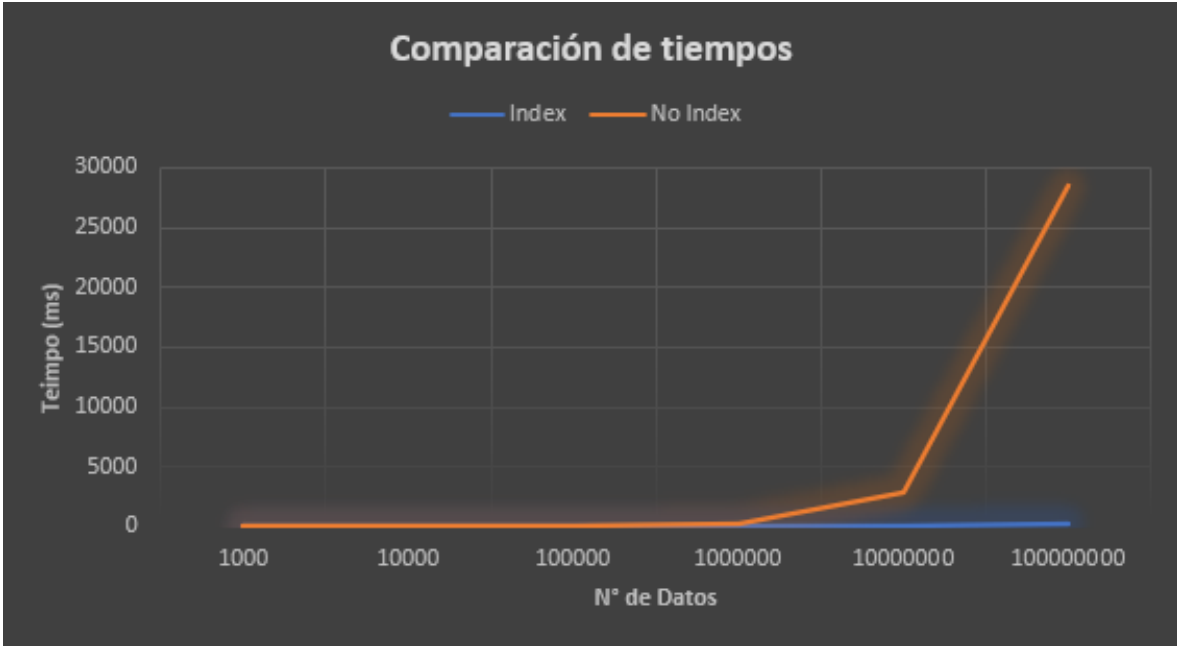


Figure 5: Gráfica de rendimiento

3 Full-Text search on Films

En esta sección, utilizaremos la función de Postgres llamadas **ts_tovector** que es útil para la búsqueda de texto completo. En el contexto de la query, se realiza lo siguiente:

- Primero se crea vector de búsqueda de texto en formato tsvector utilizando el diccionario english. La función `setweight` asigna un peso de 'A' y 'B' respectivamente al vector de búsqueda de texto generado a partir de la columna `title` y `description`. Esto indica la relevancia de cada atributo en la búsqueda a realizar. Finalmente, se concatenan ambos vectores usando el comando `'||'`.
- Luego, se agrega una nueva columna `indexado` a la tabla `film` y actualiza los valores de esa columna utilizando un vector de búsqueda de texto ponderado generado a partir de las columnas `title` y `description` de la misma tabla.
- Se crea el index GIN en el atributo `indexado` para con esto proceder a realizar las consultas requeridas.

```

10 UPDATE film SET indexed = T.indexed FROM (
11     SELECT film_id,
12         setweight(to_tsvector('english', title), 'A') || setweight(to_tsvector('english', description), 'B') AS indexed
13     FROM film
14 ) AS T WHERE film.film_id = T.film_id;
15
16 CREATE INDEX film_gin_indexed ON film using gin(indexed);

```

Figure 6: Create new attribute in table **film**

```

cs2042db2.public> CREATE INDEX film_gin_indexed on film using gin(indexed)
[2023-05-28 15:50:37] completed in 24 ms

```

Figure 7: **GIN** over **indexed** attribute

```

18 EXPLAIN ANALYZE
19 SELECT title, description FROM film WHERE description ILIKE '%man%' OR description ILIKE '%woman%';

```

Output Result 66 x

5 rows

QUERY PLAN

1	Seq Scan on film (cost=10000000000.00..10000000146.00 rows=455 width=109) (actual time=0.084..2.598 rows=318 loops=1)
2	Filter: ((description ~* '%man% '::text) OR (description ~* '%woman% '::text))
3	Rows Removed by Filter: 682
4	Planning Time: 0.396 ms
5	Execution Time: 2.625 ms

Figure 8: Query execution plan for **Query 1**

```

21 SET enable_seqscan = OFF;
22 EXPLAIN ANALYZE
23 SELECT title, description FROM film WHERE indexed @@ to_tsquery('english', 'Man | Woman');
24 -- @@ operator returns boolean: tsvector matches tsquery?

```

Output Result 69-2 x

7 rows

QUERY PLAN

1	Bitmap Heap Scan on film (cost=13.84..150.51 rows=238 width=109) (actual time=0.130..0.267 rows=239 loops=1)
2	Recheck Cond: (indexed @@ 'man' 'woman'::tsquery)
3	Heap Blocks: exact=76
4	-> Bitmap Index Scan on film_gin_indexed (cost=0.00..13.78 rows=238 width=0) (actual time=0.098..0.099 rows=239 loops=1)
5	Index Cond: (indexed @@ 'man' 'woman'::tsquery)
6	Planning Time: 0.271 ms
7	Execution Time: 0.318 ms

Figure 9: Query execution plan for **Query 2**

En esta prueba del experimento es importante ejecutar la **línea 21** puesto que, al inhabilitar el **Sequential Scan**, forzamos a PostgreSQL a hacer uso del índice creado anteriormente (**Figure 7**).

Podemos darnos cuenta que las mejoras en el tiempo de ejecución de la **Query 2** son bastante evidentes respecto a la **Query 1**, puesto que hace uso del **GIN** aplicado sobre el atributo **indexed**. Este atributo es de tipo **tsvector**, y en cada uno de sus registros contiene los lexemas y las posiciones donde inciden dichos lexemas de las palabras que aparecen en el documento.

3.1 Top K documentos

Para este ejercicio, usamos los 16,32,64 y 128 K de documentos.

```
cs2042db2=# EXPLAIN ANALYSE
SELECT title, description, ts_rank_cd(indexed, query) as rank
FROM film, to_tsquery('english', 'Man | Woman') query
ORDER BY rank DESC LIMIT 16;

EXPLAIN ANALYSE
SELECT title, description, ts_rank_cd(indexed, query) as rank
FROM film, to_tsquery('english', 'Man | Woman') query
ORDER BY rank DESC LIMIT 32;

EXPLAIN ANALYSE
SELECT title, description, ts_rank_cd(indexed, query) as rank
FROM film, to_tsquery('english', 'Man | Woman') query
ORDER BY rank DESC LIMIT 64;

EXPLAIN ANALYSE
SELECT title, description, ts_rank_cd(indexed, query) as rank
FROM film, to_tsquery('english', 'Man | Woman') query
ORDER BY rank DESC LIMIT 128;
```

Figure 10: Query sin usar índice GIN

```
QUERY PLAN
-----
Limit  (cost=168.50..168.54 rows=16 width=113) (actual time=6.659..6.662 rows=16 loops=1)
-> Sort  (cost=168.50..171.00 rows=1000 width=113) (actual time=6.656..6.657 rows=16 loops=1)
    Sort Key: (ts_rank_cd(film.indexed, ''man'' | ''woman''::tsquery)) DESC
    Sort Method: top-N heapsort  Memory: 29kB
    -> Seq Scan on film  (cost=0.00..143.50 rows=1000 width=113) (actual time=0.315..0.168 rows=1000 loops=1)
Planning Time: 4.629 ms
Execution Time: 6.708 ms
(7 rows)

QUERY PLAN
-----
Limit  (cost=173.50..173.58 rows=32 width=113) (actual time=4.008..4.013 rows=32 loops=1)
-> Sort  (cost=173.50..176.00 rows=1000 width=113) (actual time=4.008..4.010 rows=32 loops=1)
    Sort Key: (ts_rank_cd(film.indexed, ''man'' | ''woman''::tsquery)) DESC
    Sort Method: top-N heapsort  Memory: 34kB
    -> Seq Scan on film  (cost=0.00..143.50 rows=1000 width=113) (actual time=0.027..0.370 rows=1000 loops=1)
Planning Time: 0.068 ms
Execution Time: 4.027 ms
(7 rows)

QUERY PLAN
-----
Limit  (cost=178.50..178.66 rows=64 width=113) (actual time=4.011..4.020 rows=64 loops=1)
-> Sort  (cost=178.50..181.00 rows=1000 width=113) (actual time=4.011..4.014 rows=64 loops=1)
    Sort Key: (ts_rank_cd(film.indexed, ''man'' | ''woman''::tsquery)) DESC
    Sort Method: top-N heapsort  Memory: 43kB
    -> Seq Scan on film  (cost=0.00..143.50 rows=1000 width=113) (actual time=0.027..0.369 rows=1000 loops=1)
Planning Time: 0.067 ms
Execution Time: 4.035 ms
(7 rows)

QUERY PLAN
-----
Limit  (cost=183.50..183.82 rows=128 width=113) (actual time=4.029..4.044 rows=128 loops=1)
-> Sort  (cost=183.50..186.00 rows=1000 width=113) (actual time=4.028..4.034 rows=128 loops=1)
    Sort Key: (ts_rank_cd(film.indexed, ''man'' | ''woman''::tsquery)) DESC
    Sort Method: top-N heapsort  Memory: 58kB
    -> Seq Scan on film  (cost=0.00..143.50 rows=1000 width=113) (actual time=0.027..0.369 rows=1000 loops=1)
Planning Time: 0.068 ms
Execution Time: 4.061 ms
(7 rows)
```

Figure 11: Plan de ejecución de cada documento sin índice

```

cs2042db2=# EXPLAIN ANALYSE
SELECT title, description, ts_rank_cd(indexed, query) as rank
FROM film, to_tsquery('english', 'Man | Woman') query
WHERE indexed @@@ query
ORDER BY rank DESC LIMIT 16;

EXPLAIN ANALYSE
SELECT title, description, ts_rank_cd(indexed, query) as rank
FROM film, to_tsquery('english', 'Man | Woman') query
WHERE indexed @@@ query
ORDER BY rank DESC LIMIT 32;

EXPLAIN ANALYSE
SELECT title, description, ts_rank_cd(indexed, query) as rank
FROM film, to_tsquery('english', 'Man | Woman') query
WHERE indexed @@@ query
ORDER BY rank DESC LIMIT 64;

EXPLAIN ANALYSE
SELECT title, description, ts_rank_cd(indexed, query) as rank
FROM film, to_tsquery('english', 'Man | Woman') query
WHERE indexed @@@ query
ORDER BY rank DESC LIMIT 128;

```

Figure 12: Query usando índice GIN

```

----- QUERY PLAN -----
Limit  (cost=150.05..150.09 rows=16 width=113) (actual time=5.086..5.095 rows=16 loops=1)
-> Sort  (cost=150.05..150.64 rows=238 width=113) (actual time=5.082..5.086 rows=16 loops=1)
    Sort Key: (ts_rank_cd(film.indexed, ''man'' | ''woman''::tsquery)) DESC
    Sort Method: top-N heapsort  Memory: 29kB
    -> Seq Scan on film  (cost=0.00..144.09 rows=238 width=113) (actual time=0.194..4.791 rows=239 loops=1)
        Filter: (indexed @@@ ''man'' | ''woman''::tsquery)
        Rows Removed by Filter: 761
Planning Time: 0.668 ms
Execution Time: 5.152 ms
(9 rows)

----- QUERY PLAN -----
Limit  (cost=151.24..151.32 rows=32 width=113) (actual time=2.888..2.896 rows=32 loops=1)
-> Sort  (cost=151.24..151.83 rows=238 width=113) (actual time=2.886..2.889 rows=32 loops=1)
    Sort Key: (ts_rank_cd(film.indexed, ''man'' | ''woman''::tsquery)) DESC
    Sort Method: top-N heapsort  Memory: 32kB
    -> Seq Scan on film  (cost=0.00..144.09 rows=238 width=113) (actual time=0.104..2.712 rows=239 loops=1)
        Filter: (indexed @@@ ''man'' | ''woman''::tsquery)
        Rows Removed by Filter: 761
Planning Time: 0.410 ms
Execution Time: 2.927 ms
(9 rows)

----- QUERY PLAN -----
Limit  (cost=152.43..152.59 rows=64 width=113) (actual time=1.783..1.792 rows=64 loops=1)
-> Sort  (cost=152.43..153.02 rows=238 width=113) (actual time=1.782..1.785 rows=64 loops=1)
    Sort Key: (ts_rank_cd(film.indexed, ''man'' | ''woman''::tsquery)) DESC
    Sort Method: top-N heapsort  Memory: 40kB
    -> Seq Scan on film  (cost=0.00..144.09 rows=238 width=113) (actual time=0.058..1.668 rows=239 loops=1)
        Filter: (indexed @@@ ''man'' | ''woman''::tsquery)
        Rows Removed by Filter: 761
Planning Time: 0.225 ms
Execution Time: 1.812 ms
(9 rows)

----- QUERY PLAN -----
Limit  (cost=153.49..153.81 rows=128 width=113) (actual time=1.346..1.363 rows=128 loops=1)
-> Sort  (cost=153.49..154.08 rows=238 width=113) (actual time=1.345..1.352 rows=128 loops=1)
    Sort Key: (ts_rank_cd(film.indexed, ''man'' | ''woman''::tsquery)) DESC
    Sort Method: quicksort  Memory: 74kB
    -> Seq Scan on film  (cost=0.00..144.09 rows=238 width=113) (actual time=0.056..1.264 rows=239 loops=1)
        Filter: (indexed @@@ ''man'' | ''woman''::tsquery)
        Rows Removed by Filter: 761
Planning Time: 0.159 ms
Execution Time: 1.381 ms
(9 rows)

```

Figure 13: Plan de ejecución de cada documento con índice

Top K documentos	Tiempo de ejecucion con índice	Tiempo de ejecucion sin índice
16K	5.152 ms	6.708 ms
32K	2.927 ms	4.027 ms
64K	1.812 ms	4.035 ms
128K	1.381 ms	4.061 ms

Table 3: Tabla de comparación del tiempo de ejecución de cada query.

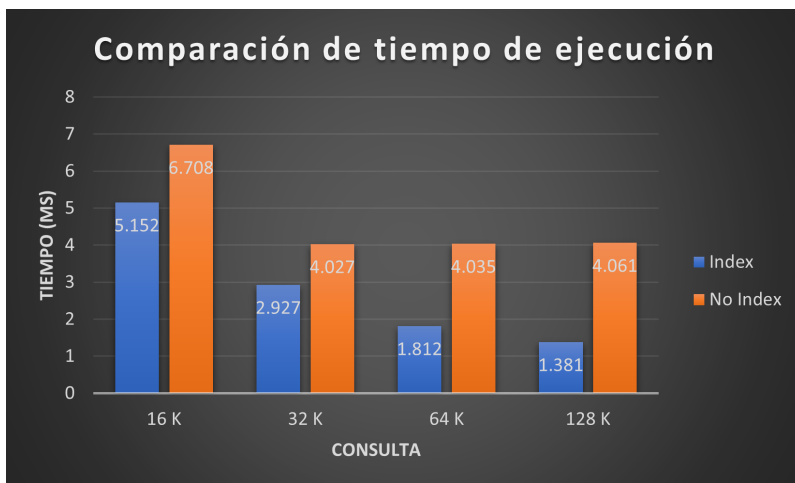


Figure 14: Comparación de tiempo de ejecución

En ambas consultas podemos apreciar que se esta utilizando un **Top-N heapsort**, el cual consiste en realizar solamente **N** pasos del algoritmo **heapsort** con la finalidad de obtener esos **N** elementos de forma ordenada. Si bien ambas consultas tiene practicamente el mismo tiempo de ejecucion, podemos evidenciar que la **Query Top-32** requiere mas memoria que la **Query Top-16**.

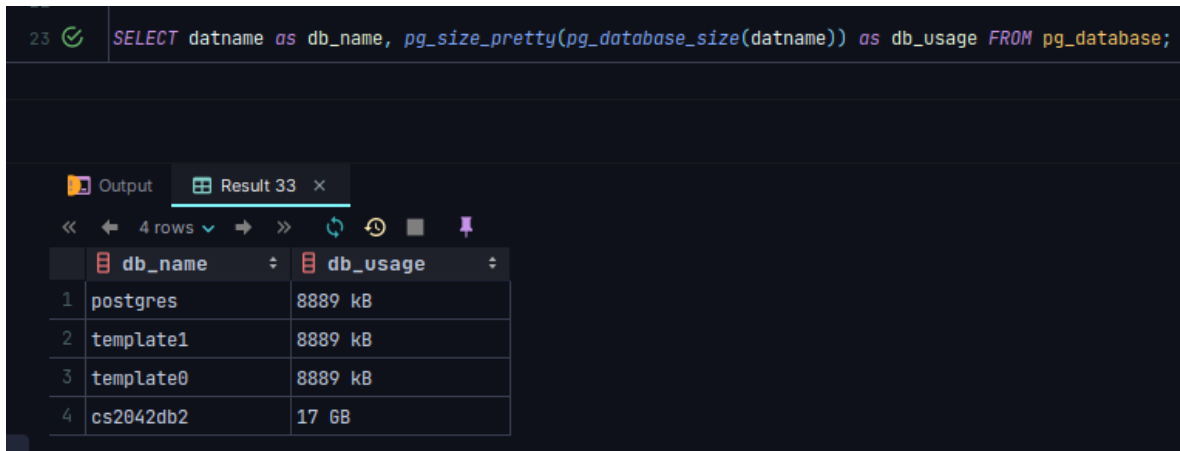
4 Full-Text search on News

En primer lugar, debemos cargar el dataset de la pagina [kaggle.com](https://www.kaggle.com) que trata sobre **articles**. Los datos se encuentran en formato **csv**, por lo que lo primero que debemos hacer es cargarlos al cliente de **PostgreSQL**.

```
cs2042db2=# \copy articles3_csv(n, id, title, publication, author, date, year, month, url, content) FROM '/tmp/articles3.csv' DELIMITER ',' CSV HEADER;
COPY 42571
cs2042db2=#
```

Figure 15: Carga de datos

Podemos apreciar que los **4251** nuevos registros se han cargado correctamente a la tabla.



The screenshot shows a PostgreSQL client interface. At the top, a query is entered: `SELECT datname as db_name, pg_size_pretty(pg_database_size(datname)) as db_usage FROM pg_database;`. Below the query, the results are displayed in a table with two columns: `db_name` and `db_usage`. The table contains four rows of data.

	db_name	db_usage
1	postgres	8889 kB
2	template1	8889 kB
3	template0	8889 kB
4	cs2042db2	17 GB

Figure 16: Visualizacion de uso de memoria hasta el momento

Hasta este momento, podemos apreciar que la base de datos **cs2042db2** ha hecho uso de **17 GB** de memoria en disco (**D: que terrible!**). Sin embargo, debemos considerar la magnitud de los datos que estamos utilizando, puesto que en el ejercicio 1 creamos un **GIN** sobre una tabla de 100 millones de registros y la computadora soporto las consultas sobre esa tabla. Esto demuestra la escalabilidad de los indices para consultas que normalmente demorarian mucho tiempo, pero ahora ponemos en evidencia el costo en memoria a pagar.

```

postgres@fedora:~
cs2042db2=# EXPLAIN ANALYSE SELECT id, title, content, ts_rank_cd(indexed, query) AS rank
FROM articles3_csv, to_tsquery('english', 'star & wars') query
WHERE articles3_csv.indexed IS NOT NULL AND articles3_csv.indexed @> query
ORDER BY rank DESC LIMIT 10;

QUERY PLAN
-----
Limit  (cost=2061.45..2061.48 rows=10 width=224) (actual time=24.397..24.401 rows=10 loops=1)
-> Sort  (cost=2061.45..2063.33 rows=751 width=224) (actual time=24.395..24.397 rows=10 loops=1)
    Sort Key: (ts_rank_cd(articles3_csv.indexed, ''star'' & ''war''::tsquery)) DESC
    Sort Method: top-N heapsort  Memory: 29kB
-> Bitmap Heap Scan on articles3_csv  (cost=33.82..2045.22 rows=751 width=224) (actual time=2.117..24.041 rows=950 loops=1)
    Recheck Cond: (indexed @> ''star'' & ''war''::tsquery)
    Heap Blocks: exact=731
-> Bitmap Index Scan on articles3_csv_gin  (cost=0.00..33.63 rows=751 width=0) (actual time=1.663..1.663 rows=950 loops=1)
    Index Cond: (indexed @> ''star'' & ''war''::tsquery)

Planning Time: 6.393 ms
Execution Time: 24.504 ms
(11 rows)

cs2042db2=# EXPLAIN ANALYSE SELECT id, title, content, ts_rank_cd(indexed, query) AS rank
FROM articles3_csv, to_tsquery('english', 'star & wars') query
WHERE articles3_csv.indexed IS NOT NULL
ORDER BY rank DESC LIMIT 10;

QUERY PLAN
-----
Limit  (cost=6333.08..6333.11 rows=10 width=224) (actual time=427.777..427.781 rows=10 loops=1)
-> Sort  (cost=6333.08..6439.51 rows=42571 width=224) (actual time=427.774..427.776 rows=10 loops=1)
    Sort Key: (ts_rank_cd(articles3_csv.indexed, ''star'' & ''war''::tsquery)) DESC
    Sort Method: top-N heapsort  Memory: 28kB
-> Seq Scan on articles3_csv  (cost=0.00..5413.14 rows=42571 width=224) (actual time=0.209..415.275 rows=42570 loops=1)
    Filter: (indexed IS NOT NULL)
    Rows Removed by Filter: 1

Planning Time: 0.284 ms
Execution Time: 427.829 ms
(9 rows)

```

Figure 17: Query with and without index

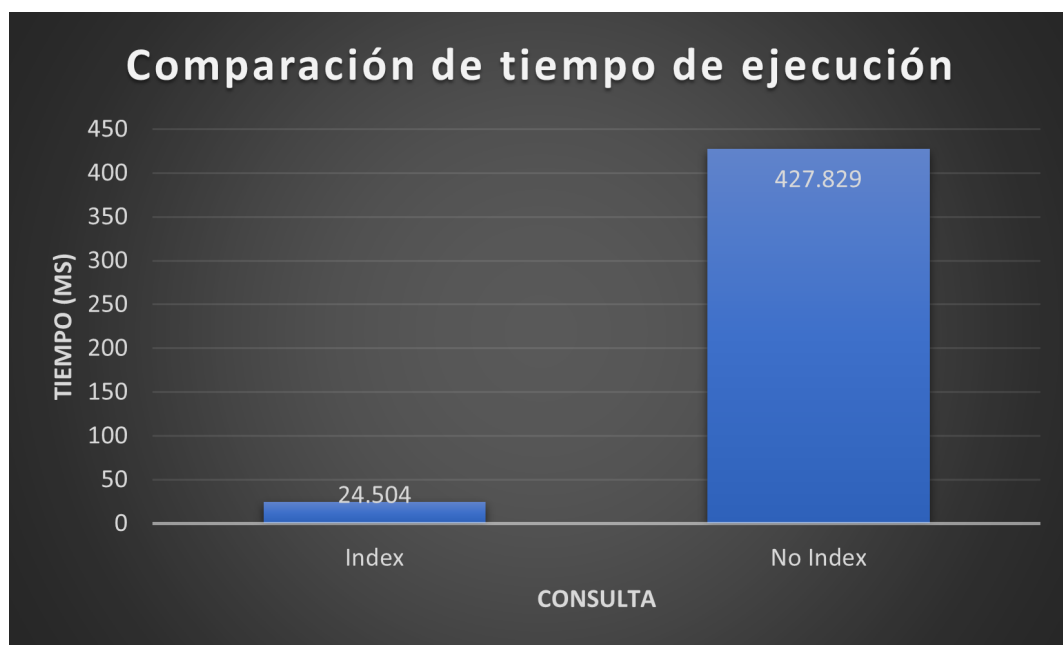


Figure 18: Comparación del tiempo de ejecución

Como podemos apreciar en la **Query 1**, se hace uso del índice para filtrar aquellos **indexed** que hacen match con la **query** para poder realizar la búsqueda en el índice, la cual es bastante rápida. Cuando no se hace uso del índice **Query 2** no se verifica el índice, por lo que la consulta se ralentiza. Aquí se puede apreciar bastante el poder del **GIN**, lo que justifica su costo en disco.

	id	title	content	rank
1	195451	Star Wars has always been political. Here's why the alt-right is claiming otherwise.	All art is political, but science fiction is perhaps the most conscientiously political genre.	14.845985
2	217788	If it's going to work, 'Star Wars' needs to get past the Rebellion and the Empire	This post discusses the plot of "Rogue One," which I reviewed last week, and much other ...	9.272831
3	188835	Bobbleheads and droids lure collectors to Disney's 'Star Wars' event	For "Star Wars" collectors, this weekend's fan convention in Orlando, Florida, is a ... even.	6.11499
4	198882	I'm a Star Wars newbie. Here's what I figured out based only on The Force Awakens.	If you're celebrating Star Wars Day today, you might be choosing your own Lightsaber color o...	5.992811
5	283678	Rogue One review: this is the first Star Wars movie to acknowledge the whole franchise is abo...	People die in Rogue One: A Star Wars Story. A lot of them. They die horrible deaths in sp...	5.918431
6	217650	'Rogue One' doesn't offer much joy, but Star Wars fans will enjoy it anyway	A movie has been made for "Star Wars" fans that finally answers many of the question...	5.578834
7	199327	Oscars 2016: predictions for all 24 Academy Award categories	The 88th annual Academy Awards will begin at 8:38 pm Eastern on Sunday, February 28, broadca...	5.4593586
8	283950	Carrie Fisher's legacy goes far beyond Star Wars: 9 things to read and watch	Carrie Fisher, who died December 27 at the age of 68, was best known for her iconic, kickass...	4.784792
9	195218	Carrie Fisher, 'Star Wars' Princess Leia, dies at 68	Carrie Fisher, who rose to fame as Princess Leia in the "Star Wars" films and later endured ...	4.6953586
10	199873	Finally, the first trailer for the new Star Wars movie is here	This morning, Star Wars fans who tuned in to ABC's Good Morning America got their first glim...	4.1784392

Figure 19: Query **Top-10** result

Finalmente, estos serian los **Top-10** documentos que hablan de **Star Wars** obtenidos con el uso del **GIN** ordenados por su relevancia.

5 Conclusiones

- En el primer ejercicio, pudimos observar que efectivamente, el uso del índice **GIN** reduce el tiempo de ejecución de una query al tratarse de un database muy grande. En tablas pequeñas, su uso puede ser nulo o inclusive peor que si no la usamos.
- Aprendimos cómo y para qué se usa la función **to.tsvector** al momento de aplicar índice **GIN** en las consultas realizadas.
- Concluimos que el tiempo de ejecución para la query con índice **GIN** en el tercer ejercicio reducía el tiempo de búsqueda en gran escala pero a cambio tiene un gran costo de memoria.



Figure 20: Gracias por su atención