Ranking Retrieval with PostgreSQL

Profesor Heider Sanchez

El objetivo de este laboratorio es poner a prueba las técnicas de indexación de textos en PostgreSQL (full-text search index) mediante tres experimentos.

P1. Sequential Scan vs GIN:

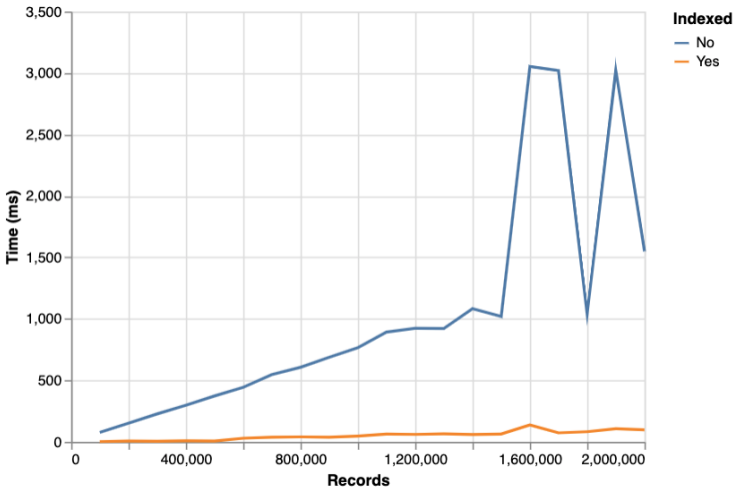
El primer experimento consiste en probar el índice invertido GIN representando el texto con **trigramas**. Un trigrama es un grupo de tres caracteres consecutivos tomados de una cadena. Ejemplo, los trigramas de la palabra “amor” son “amo” y “mor”. Indexar un atributo tipo texto con trigramas es eficaz en la mayoría de lenguajes naturales mejorando considerablemente las búsquedas textuales.

https://www.postgresql.org/docs/13/pgtrgm.html

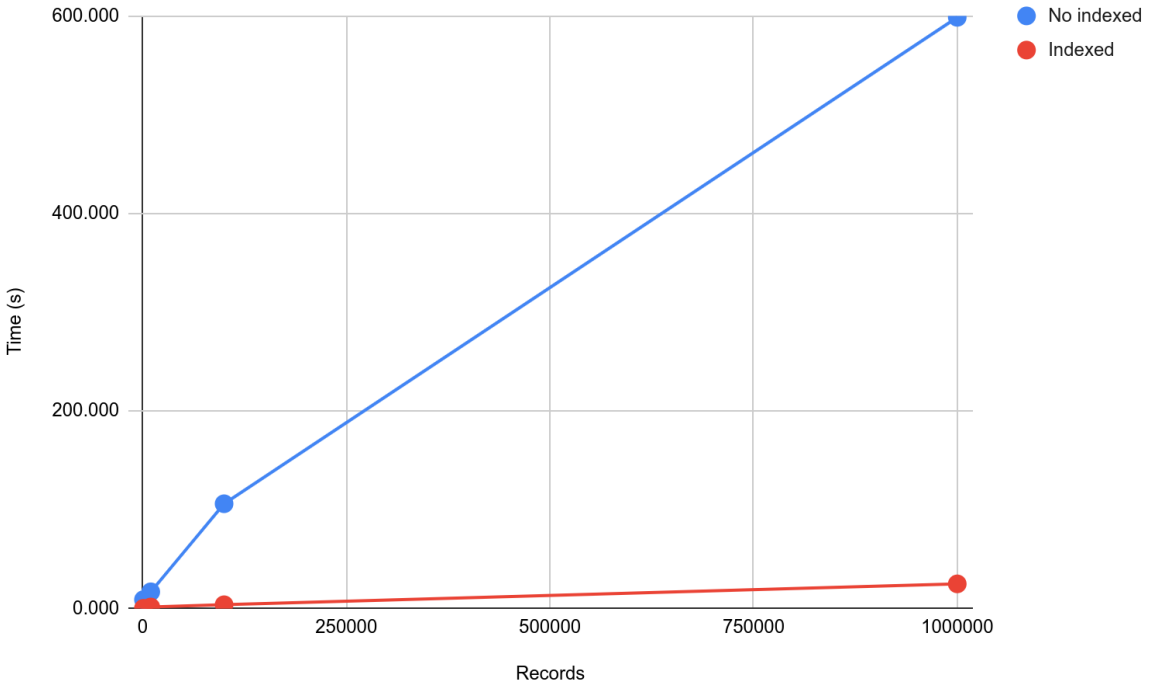
Tomando como base el script dato en clase, se le pide realizar lo siguiente:

- Crear una tabla con dos atributos textuales, uno sin indexar y el otro indexado.
- Llenar datos aleatorios para diferentes cantidades.
- Ejecutar consultas sobre ambos atributos y tomar los tiempos

Mostrar el plan de ejecución y un gráfico como resultado de la experimentación (ver gráfico de referencia).



	Time (ms)	
Records	**No indexed**	**Indexed**
1000	9.079	0.342
10000	16.996	1.616
100000	106.164	3.927
1000000	598.555	25.064



P2. Full-text search on Films

El segundo experimento consiste en aplicar el índice invertido GIN sobre los atributos textuales de la tabla “film” (dvdrental).

- Restaurar la base de datos en su servidor PostgreSQL
- Crear un nuevo atributo indexado compuesto por el titulo y la descripción de la película.
  - El tipo de dato corresponde al vector de pesos de los términos
- Ejecutar consultas sobre los atributos sin indexar y sobre el atributo indexado
  - Tomar los tiempos para diferentes rankings (top k)

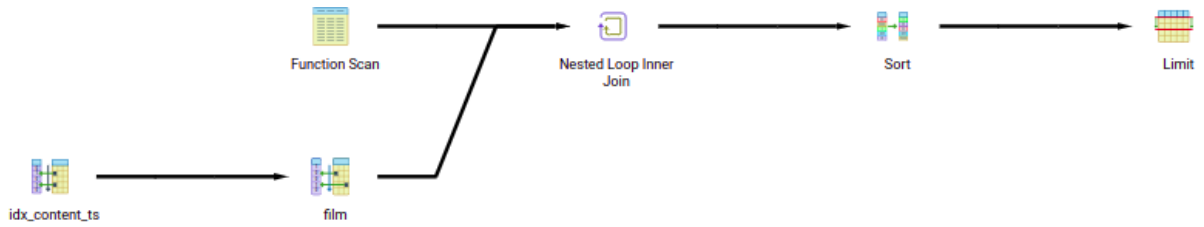
Mostrar el plan de ejecución y un gráfico como resultado de la experimentación

Query:

```
--TOP K
--Ranking (1): 1.638 ms
--Ranking (10): 1.969 ms
--Ranking (100): 2.054 ms
EXPLAIN ANALYZE
SELECT title, description,
       ts_rank_cd(content_ts, query_ts) AS score
FROM film, to_tsquery('english', 'man | woman') query_ts
WHERE query_ts @@ content_ts
ORDER BY score desc
LIMIT 1;
```

Plan de ejecución:

	QUERY PLAN text
1	Limit (cost=25.26..25.26 rows=1 width=113) (actual time=1.580..1.583 rows=1 loops=1)
2	-> Sort (cost=25.26..25.27 rows=5 width=113) (actual time=1.578..1.581 rows=1 loops=1)
3	Sort Key: (ts_rank_cd(film.content_ts, query_ts.query_ts)) DESC
4	Sort Method: top-N heapsort Memory: 25kB
5	-> Nested Loop (cost=8.04..25.23 rows=5 width=113) (actual time=0.111..1.485 rows=239 loops=1)
6	-> Function Scan on query_ts (cost=0.00..0.01 rows=1 width=32) (actual time=0.004..0.007 rows=1 loops=1)
7	-> Bitmap Heap Scan on film (cost=8.04..25.16 rows=5 width=286) (actual time=0.087..0.270 rows=239 loops=1)
8	Recheck Cond: (query_ts.query_ts @@ content_ts)
9	Heap Blocks: exact=73
10	-> Bitmap Index Scan on idx_content_ts (cost=0.00..8.04 rows=5 width=0) (actual time=0.068..0.069 rows=239 loops=1)
11	Index Cond: (content_ts @@ query_ts.query_ts)
12	Planning Time: 0.207 ms
13	Execution Time: 1.638 ms



P3. Full-text search on News

El tercer experimento consiste en aplicar el índice invertido GIN sobre los atributos textuales de la tabla “articles” (all the news).

- Crear la tabla Articles y llenar los datos desde los archivos CSV
- Crear un nuevo atributo indexado compuesto por el titulo y el contenido de la noticia.
  - El tipo de dato corresponde al vector de pesos de los términos
- Ejecutar consultas sobre los atributos sin indexar y sobre el atributo indexado
  - Tomar los tiempos para diferentes rankings (top k)

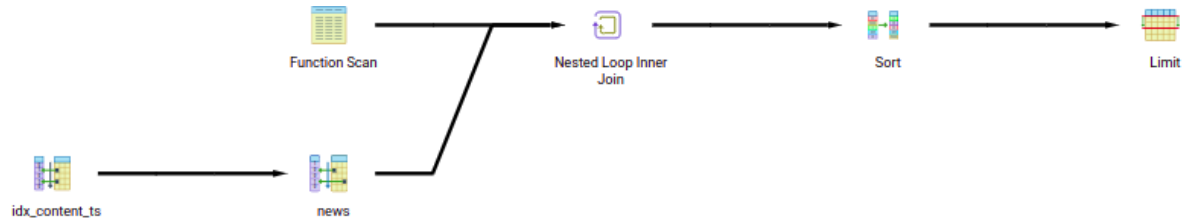
Mostrar el plan de ejecución y un gráfico como resultado de la experimentación

Query 1:

```
--Ranking (1): 381.465 ms
--Ranking (10): 384.137 ms
--Ranking (100): 392.058 ms
EXPLAIN ANALYZE
SELECT title, content,
       ts_rank_cd(content_ts, query_ts) AS score
FROM news, to_tsquery('english', 'Obama | Trump') query_ts
WHERE query_ts @@ content_ts
ORDER BY score desc
LIMIT 10;
```

Plan de ejecución:

	QUERY PLAN text
1	Limit (cost=900.86..900.89 rows=10 width=618) (actual time=384.080..384.084 rows=10 loops=1)
2	-> Sort (cost=900.86..901.49 rows=250 width=618) (actual time=384.078..384.081 rows=10 loops=1)
3	Sort Key: (ts_rank_cd(news.content_ts, query_ts.query_ts)) DESC
4	Sort Method: top-N heapsort Memory: 27kB
5	-> Nested Loop (cost=17.94..895.46 rows=250 width=618) (actual time=16.707..372.236 rows=23390 loops=1)
6	-> Function Scan on query_ts (cost=0.00..0.01 rows=1 width=32) (actual time=0.006..0.007 rows=1 loops=1)
7	-> Bitmap Heap Scan on news (cost=17.94..892.32 rows=250 width=707) (actual time=16.570..63.828 rows=23390 loops=1)
8	Recheck Cond: (query_ts.query_ts @@ content_ts)
9	Heap Blocks: exact=6125
10	-> Bitmap Index Scan on idx_content_ts (cost=0.00..17.88 rows=250 width=0) (actual time=13.769..13.770 rows=23390 loops=1)
11	Index Cond: (content_ts @@ query_ts.query_ts)
12	Planning Time: 0.305 ms
13	Execution Time: 384.137 ms



Query 2:

```
--Ranking (1): 30.145 ms
--Ranking (10): 31.558 ms
--Ranking (100): 36.297 ms
EXPLAIN ANALYZE
SELECT title, content,
       ts_rank_cd(content_ts, query_ts) AS score
FROM news, plainto_tsquery('english', 'Independency of United States') query_ts
WHERE query_ts @@ content_ts
ORDER BY score desc
LIMIT 10;
```

	QUERY PLAN text
1	Limit (cost=900.86..900.89 rows=10 width=618) (actual time=31.506..31.510 rows=10 loops=1)
2	-> Sort (cost=900.86..901.49 rows=250 width=618) (actual time=31.505..31.507 rows=10 loops=1)
3	Sort Key: (ts_rank_cd(news.content_ts, query_ts.query_ts)) DESC
4	Sort Method: top-N heapsort Memory: 28kB
5	-> Nested Loop (cost=17.94..895.46 rows=250 width=618) (actual time=2.093..30.743 rows=1397 loops=1)
6	-> Function Scan on query_ts (cost=0.00..0.01 rows=1 width=32) (actual time=0.004..0.005 rows=1 loops=1)
7	-> Bitmap Heap Scan on news (cost=17.94..892.32 rows=250 width=707) (actual time=2.023..3.932 rows=1397 loops=1)
8	Recheck Cond: (query_ts.query_ts @@ content_ts)
9	Heap Blocks: exact=1131
10	-> Bitmap Index Scan on idx_content_ts (cost=0.00..17.88 rows=250 width=0) (actual time=1.816..1.816 rows=1397 loops=1)
11	Index Cond: (content_ts @@ query_ts.query_ts)
12	Planning Time: 0.445 ms
13	Execution Time: 31.558 ms