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:

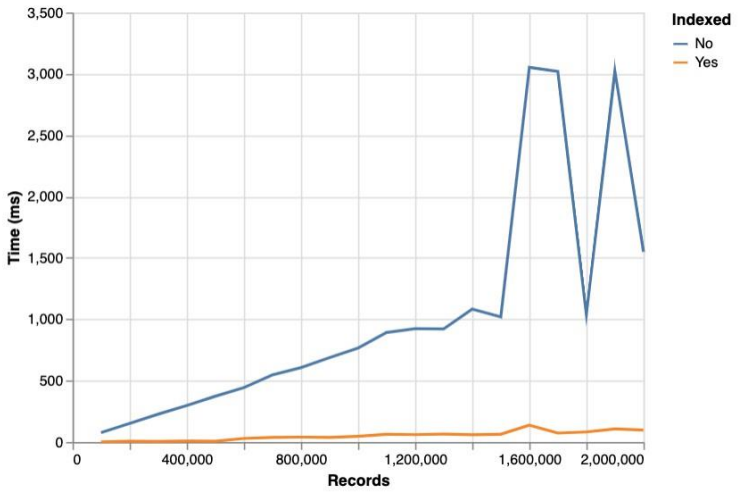
El primer experimento consiste en probar el índice invertido GIN representando el texto con **trigramas**. Un trigrama es un grupo de tres caracteres consecutivos tomados de una cadena. Ejemplo, los trigramas de la palabra "amor" son "amo" y "mor". Indexar un atributo tipo texto con trigramas es eficaz 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).

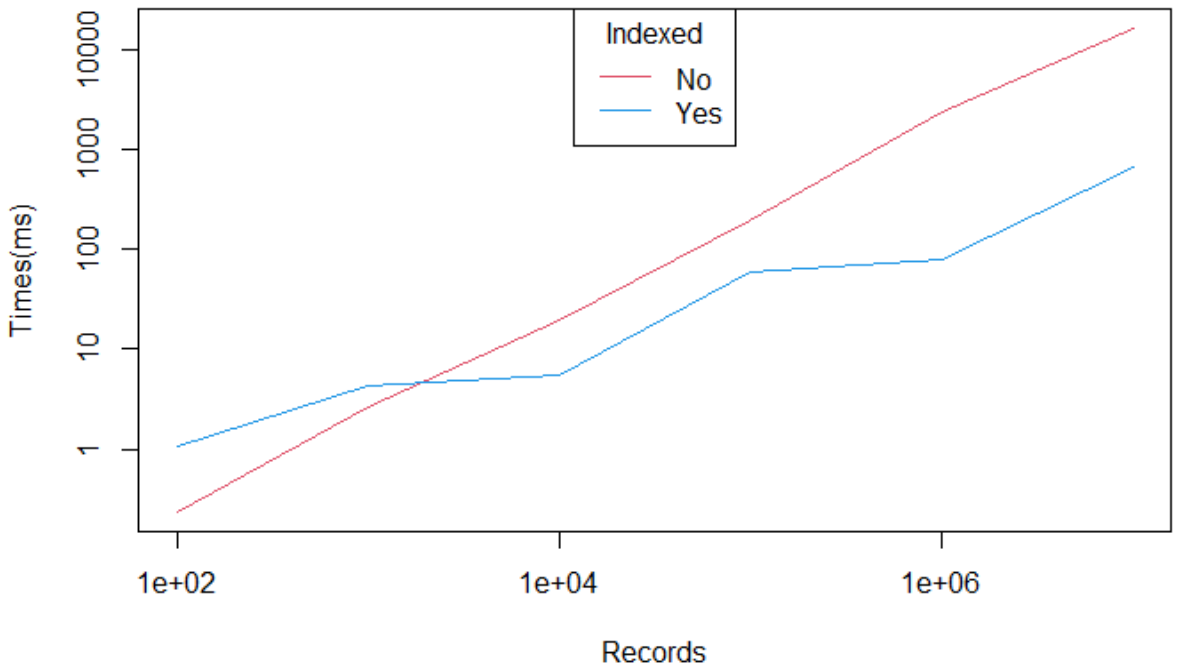



```
1.   CREATE TABLE articles (
2.    body text,
3.    body_indexed text
4.  );
5.
6.  -- create extension
7.  CREATE EXTENSION pg_trgm;
8.
9.  -- create an index
10. CREATE INDEX articles_search_idx ON articles USING gin (body_indexed gin_trgm_ops);
11.
12. -- populate table with data
13. delete from articles;
14. INSERT INTO articles
15.   SELECT
16.   md5(random()::text),
17.   md5(random()::text)
18.   from (
19.            SELECT * FROM generate_series(1,1000000) AS id
20.   ) as T
21.
22. -- show data
```

```
23. select * from articles limit 20
24.
25. -- query
26. explain analyze
27. select * from articles where body ilike '%abc%';
28. -- 100          1000  10000     100000       1000000             10000000
29. -- 0,236    2,680     20,003    198,095      2364,945            16798,057
30.
31. explain analyze
32. select * from articles where body_indexed ilike '%abc%';
33. -- 100          1000  10000     100000       1000000             10000000
34. -- 1,052    4,375     5,571     60,685       78,206              660,443
```

Plan de ejecución

Gráfico de resultados



P2. Full-text search on Films

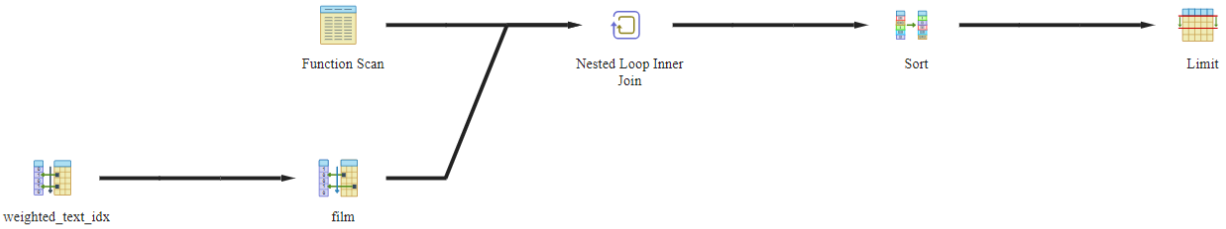
El segundo experimento consiste en aplicar el índice invertido GIN sobre los atributos textuales de la tabla “film” (dvdrental).

- 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

```
1.  -- tokenization
2.  SELECT * FROM ts_parse('default','Si piensas que los usuarios de tus programas son idiotas,
    solo los idiotas usaran tus programas')
3.
4.  SELECT ts_parse('default', description) FROM film where title = 'Intolerable Intentions'
5.
6.  -- Lexxeme (stemming)
7.  select ts_lexize('spanish_stem','Programador')
8.  select ts_lexize('spanish_stem','Programación')
9.  select ts_lexize('spanish_stem','Programadores')
10.
11. -- Tokenization & Lexema
12. SELECT * FROM ts_debug('spanish','Si piensas que los usuarios de tus programas son idiotas,
    solo los idiotas usaran tus programas')
13.
14. -- Creating a document vector (term: position)
15. /*tokenization, remove stopwords, stemming*/
16. select to_tsvector(description)from film;
17. select to_tsvector(title || description)from film limit 2;
18.
19. -- Weighting
20. select setweight(to_tsvector(title), 'A') ||
21.             setweight(to_tsvector(description), 'B')
22. from film limit 10;
23.
24. /* --- Indexing --- */
25. -- create an addional column
26. alter table film add column weighted_text tsvector;
27.
28. -- Populating the document vectors
29. update film set weighted_text = T.weighted
30. from (select film_id,
31.             setweight(to_tsvector(title), 'A') ||
32.             setweight(to_tsvector(description), 'B')
33.    as weighted
34.    from film
35.    ) as T
36.    where T.film_id = film.film_id
37.
38. select film_id, title, description, weighted_text from film limit 10;
39.
40. -- Indexing
41. create index weighted_text_idx on film using GIN (weighted_text);
42.
43. -- Querying
44.
45. select plainto_tsquery('The man or the woman');
46. select to_tsquery('Man | Woman');
47.
48. explain analyze
49. select * from film where description ilike '%man%' or description ilike '%woman%';
50. -- 10.084 ms
51.
52. explain analyze
53. select * from film where to_tsquery('Man | Woman') @@ weighted_text;
54. -- 0.540 ms
55.
56. -- Optimization
57. -- query top-10
58. explain analyze
59. select title, description, ts_rank_cd(weighted_text, query) as rank
60. from film, to_tsquery('Man | Woman') query
61. where query @@ weighted_text
62. order by rank desc
63. limit 100;
64. -- 10    20    30     100
65. -- 0.845 1.010 0.671  0.687
```

Plan de ejecución



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