

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:

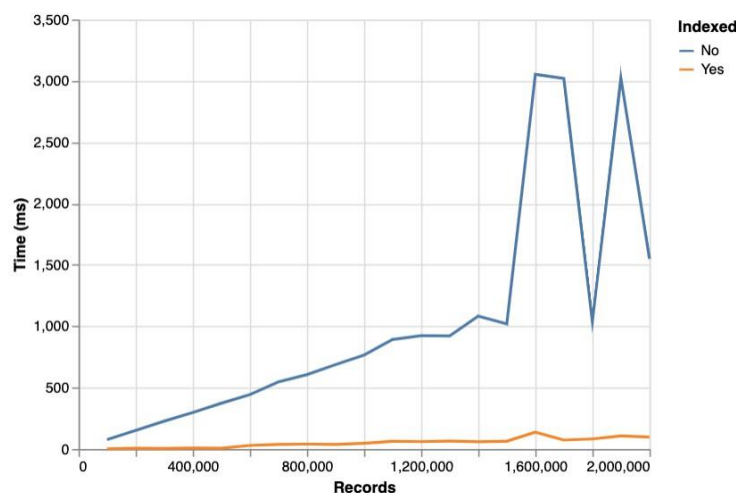
El primer experimento consiste en probar el índice invertido GIN representando el texto con **trigramas**. Un trígrama es un grupo de tres caracteres consecutivos tomados de una cadena. Ejemplo, los trigramas de la palabra “amor” son “amo” y “mor”. Indexar un atributo tipo texto con trigramas es eficaz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

C# 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%';

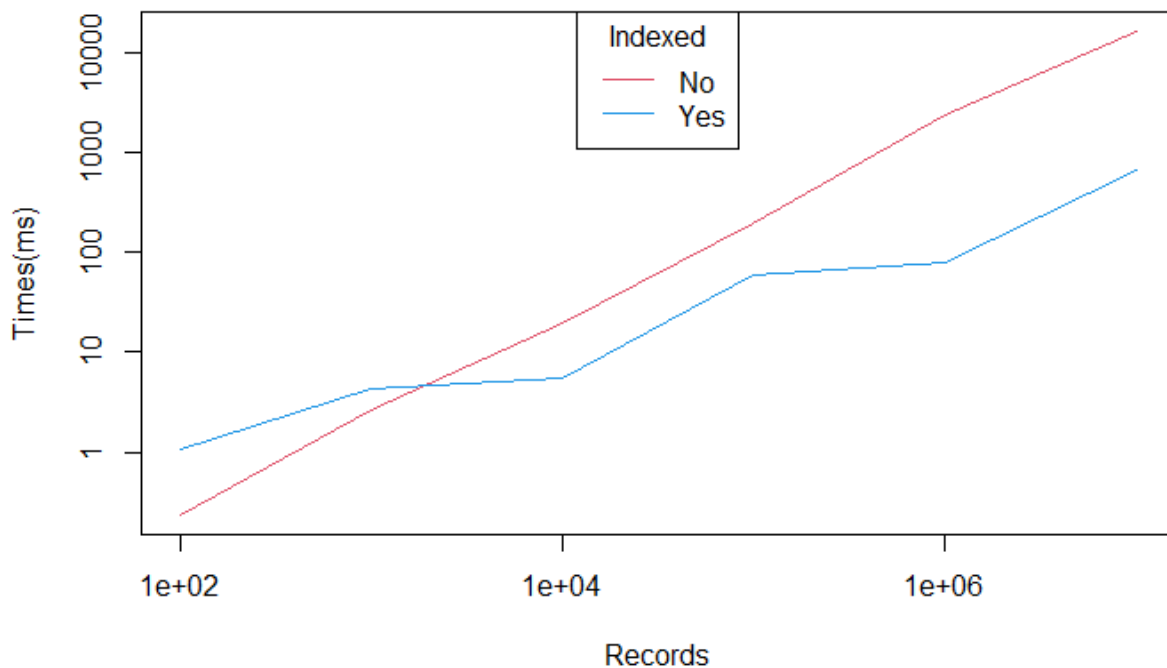
33. -- 100 1000 10000 100000 1000000 10000000

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

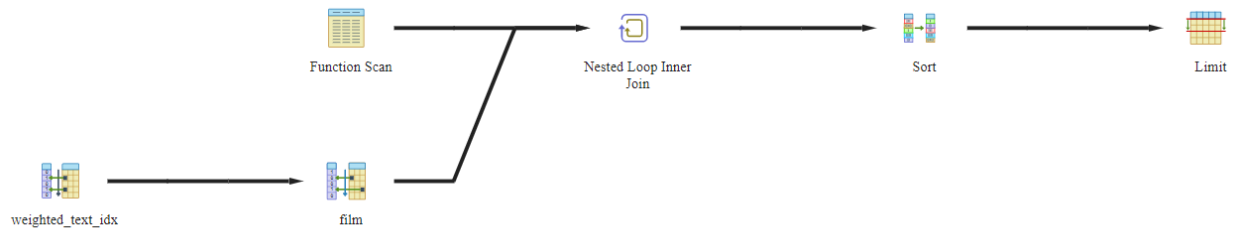
El segundo experimento consiste en aplicar el índice invertido GIN sobre los atributos textuales de la tabla "film" ([dvdrental](#)).

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

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

```
1. CREATE TABLE articles (  
2.   num decimal,  
3.   id integer,  
4.   title text,  
5.   publication char[45],  
6.   author char[45],  
7.   date date,  
8.   year date,  
9.   month date,  
10.  url text,  
11.  content text  
12. );  
13.  
14. COPY articles FROM 'C:\Users\miana\OneDrive\Escritorio\Lab8_2DB2\articles1.csv' WITH (FORMAT  
    csv);  
15.  
16. -- Creating a document vector (term: position)  
17. /*tokenization, remove stopwords, stemming*/  
18. select to_tsvector(content) from articles;  
19. select to_tsvector(title || content) from articles limit 2;  
20.  
21. -- Weighting  
22. select setweight(to_tsvector(title), 'A') ||  
23.        setweight(to_tsvector(content), 'B')  
24. from articles limit 10;  
25.  
26. /* --- Indexing --- */  
27. -- create an additional column  
28. alter table articles add column weighted_text tsvector;  
29.  
30. -- Populating the document vectors  
31. update articles set weighted_text = T.weighted  
32. from (select id,  
33.         setweight(to_tsvector(title), 'A') ||  
34.         setweight(to_tsvector(content), 'B')  
35.        as weighted  
36.       from articles  
37.      ) as T  
38. where T.id = film.id  
39.  
40. select id, title, content, weighted_text from articles limit 10;  
41.  
42. -- Indexing  
43. create index weighted_text_idx on articles using GIN (weighted_text);  
44.  
45. -- Querying  
46.  
47. select plainto_tsquery('The man or the woman');  
48. select to_tsquery('Man | Woman');  
49.  
50. explain analyze  
51. select * from articles where content ilike '%man%' or content ilike '%woman%';  
52. -- 10.084 ms  
53.  
54. explain analyze
```

```
C:\55. select * from articles where to_tsquery('Man | Woman') @@ weighted_text;  
56. -- 0.204 ms  
57.  
58. -- Optimization  
59. -- query top-100  
60. explain analyze  
61. select title, content, ts_rank_cd(weighted_text, query) as rank  
62. from articles, to_tsquery('Man | Woman') query  
63. where query @@ weighted_text  
64. order by rank desc  
65. limit 100;  
66. -- 100  
67. -- 1.034
```

Plan de ejecución

