Pokročilé databázové technológie

Zadanie 2 – vyhľadávanie a indexovanie Marek Adamovič

Cvičenie: Piatok 11:00 - 12:40

Ing. Ján Balažia, PhD.

Obsah

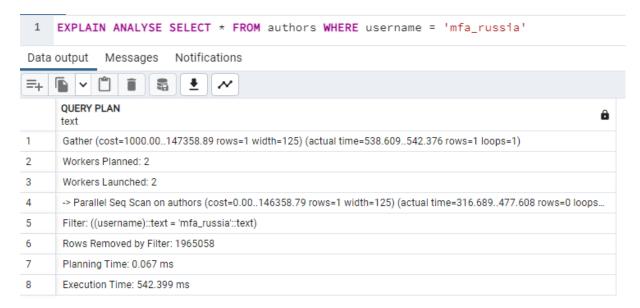
1.	otázka	3
2.	otázka	4
3.	otázka	6
4.	otázka	7
5.	otázka	8
6.	otázka	. 10
7.	otázka	. 11
8.	otázka	. 12
9.	otázka	. 14
10.	otázka	. 16
11.	otázka	. 18
12.	otázka	. 19
13.	otázka	. 21
14.	otázka	. 22
15.	otázka	. 24
16.	otázka	. 25

1. otázka

Otázka:

Vyhľadajte v authors username s presnou hodnotou 'mfa_russia' a analyzujte daný select. Akú metódu vám vybral plánovač a prečo - odôvodnite prečo sa rozhodol tak ako sa rozhodol?

Odpoveď:



Plánovač vybral paralelný sekvenčný sken, keďže vie, že potrebujeme skenovať celú tabuľku. Pri skenovaní celej tabuľky ľahko rozdelí prácu na viacerých workerov, čo značne urýchli dopyt. Plánovač nemôže použiť index scan, keďže defaultný index je len pre stĺpec s PRIMARY KEY (teda v tomto prípade id) a iný index vytvorený nemáme.

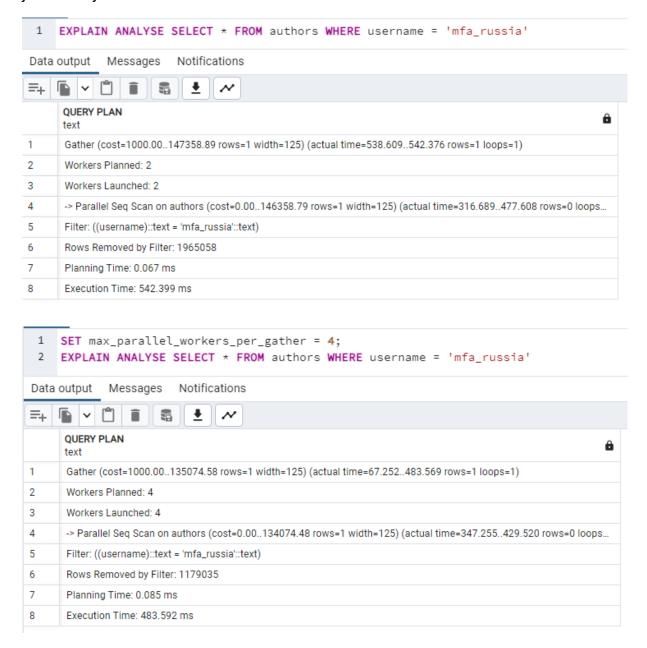
2. otázka

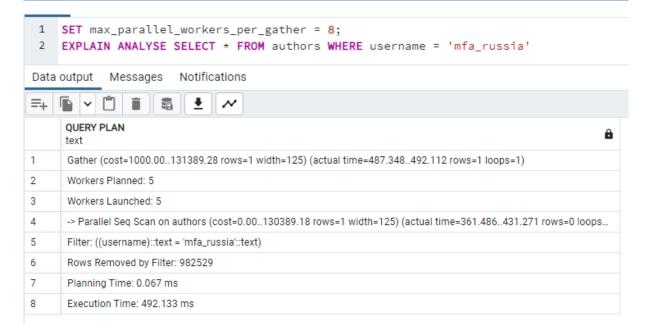
Otázka:

Koľko workerov pracovalo na danom selecte a na čo slúžia? Zdvihnite počet workerov a povedzte ako to ovplyvňuje čas. Je tam nejaký strop? Ak áno, prečo? Od čoho to závisí (napíšte a popíšte všetky parametre)?

Odpoveď:

Na danom selecte pracovali 2 workeri. Slúžia na využitie paralelizmu pri dopytoch. Každý worker dostane pridelenú robotu, ktorej výsledky sa na záver dopytu spoja do jedného výsledku.





Max. workers	Used workers	Time (in ms)
2	2	542.399
4	4	483.592
8	5	492.133

Prvým stropom je premenná max_parallel_workers_per_gather, ktorá udáva, koľko workerov môže byť použitých na jedno využitie paralelizmu. Defaultne je nastavená na hodnotu 2. Ďalším obmedzením je premenná max_parallel_workers, čo je maximálny počet pre všetky použitia paralelizmu (teda je to globálnejšie nastavenie). Defaultná hodnota tejto premennej je 8. Posledným obmedzením je premenná max_worker_processes, ktorá určuje, koľko procesov môže bežať na pozadí. Táto premenná sa dá zmeniť len počas štartu servera. Defaultná hodnota je 8.

Taktiež je nutné poznamenať, že plánovač vyberá počet workerov podľa toho, čo sa najviac oplatí. Vyšší počet workerov nemusí vždy znamenať lepší čas. Môže to byť spôsobené napríklad tým, že čím viacej workerov máme, tým viacej výsledkov sa musí na konci dopytu spájať do finálneho výstupu.

3. otázka

Otázka:

Vytvorte btree index nad username a pozrite ako sa zmenil čas a porovnajte výstup oproti požiadavke bez indexu. Potrebuje plánovač v tejto požiadavke viac workerov? Čo ovplyvnilo zásadnú zmenu času?

Odpoveď:



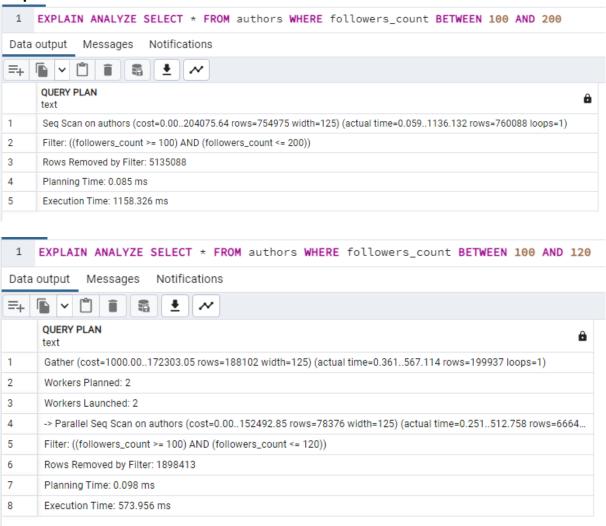
Plánovač v tomto prípade nepotrebuje viac workerov, keďže neprehľadávame tabuľku sekvenčne, ale podľa indexu (Index Scan). Zásadnú zmenu času spôsobilo, že vďaka indexu vieme, kde máme hľadať záznamy s názvom, ktorý máme v podmienke. Tým pádom nemusíme prehľadávať zbytok tabuľky. Je to podobný princíp ako keby v telefónnom zozname hľadáme konkrétne meno -> taktiež nemusíme prečítať celý telefónny zoznam. Zlepšili sme čas na úkor pamäte.

4. otázka

Otázka:

Vyberte používateľov, ktorý majú followers_count väčší, rovný ako 100 a zároveň menší, rovný 200. Potom zmeňte rozsah na väčší, rovný ako 100 a zároveň menší, rovný 120. Je tam rozdiel, ak áno prečo?

Odpoveď:



Je tam veľký rozdiel v čase kvôli tomu, že plánovač použil v druhom prípade workerov, teda paralelizmus. O počte workerov rozhoduje plánovač na základe podmienky vo WHERE klauzule. Paralelný sekvenčný sken je v tomto prípade rýchlejší ako klasický sekvenčný sken.

5. otázka

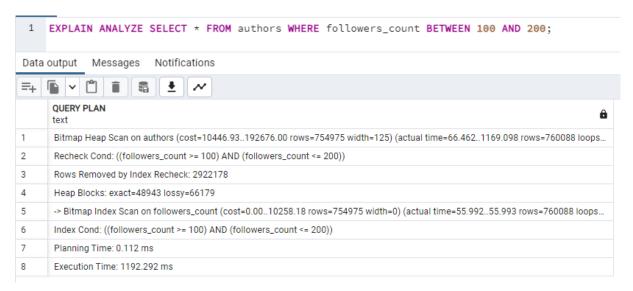
Otázka:

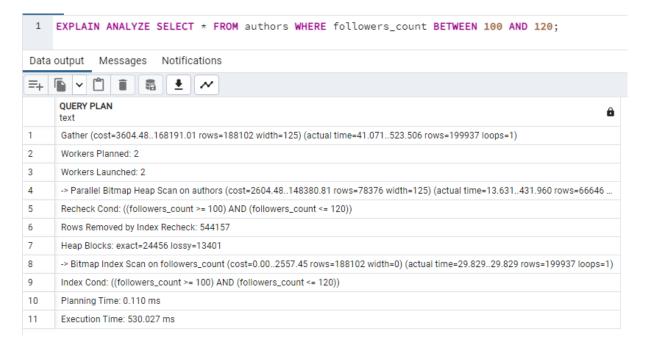
Vytvorte index nad 4 úlohou a v oboch podmienkach popíšte prácu s indexom. Čo je to Bitmap Index Scan a prečo je tam Bitmap Heap Scan? Prečo je tam recheck condition? Použil sa vždy index?

Odpoveď:



Vytvorili sme index nad stĺpcom followers_count (keďže naň sú smerované nasledujúce selecty)





V prvom prípade sa použil Bitmap Heap Scan, ktorý využíva Bitmap Index Scan. Bitmap Index Scan najskôr vytvorí bitmapu podľa indexov, aby sa zistilo, ktoré bloky dát treba prečítať. V závislosti od (ne)dostatku pamäte môžu byť v tejto bitmape presné ukazovatele priamo na dáta alebo nepresné ukazovatele na stránku, kde je dát viacej. Pri nepresných ukazovateľoch potom musíme riešiť recheck condition, ktorý nám na konkrétnych dátach znovu skontroluje danú podmienku. Túto bitmapu používa Bitmap Heap Scan, aby vedel, ktoré dáta má prečítať. Index sa použil v oboch prípadoch, avšak paralelizmus sa použil len v druhom, vďaka čomu bol druhý dopyt značne rýchlejší.

6. otázka

Otázka:

Vytvorte ďalšie 3 btree indexy na name, followers_count, a description a insertnite si svojho používateľa (to je jedno aké dáta) do authors. Koľko to trvalo? Dropnite indexy a spravte to ešte raz. Prečo je tu rozdiel?

Odpoveď:

```
INSERT INTO authors VALUES(-27, 'test', 'test', 'test', 27, 27, 27)
Data output
           Messages Notifications
INSERT 0 1
Query returned successfully in 40 secs 906 msec.
   DROP INDEX name;
 2 DROP INDEX description;
    DROP INDEX followers_count;
            Messages Notifications
Data output
DROP INDEX
Query returned successfully in 205 msec.
   INSERT INTO authors VALUES(-28, 'test', 'test', 'test', 27, 27, 27)
Data output
           Messages
                      Notifications
INSERT 0 1
Query returned successfully in 78 msec.
```

Insert bez indexov trval oveľa kratšie. To vďaka tomu, že keď vkladáme záznam a existujú indexy, musíme ho vložiť a usporiadať aj do týchto indexov, čo je značne pomalšie oproti vloženiu (len) do hlavnej tabuľky.

7. otázka

Otázka:

Vytvorte btree index nad conversations pre retweet_count a pre content. Porovnajte ich dĺžku vytvárania. Prečo je tu taký rozdiel? Čím je ovplyvnená dĺžka vytvárania indexu a prečo?

Odpoveď:



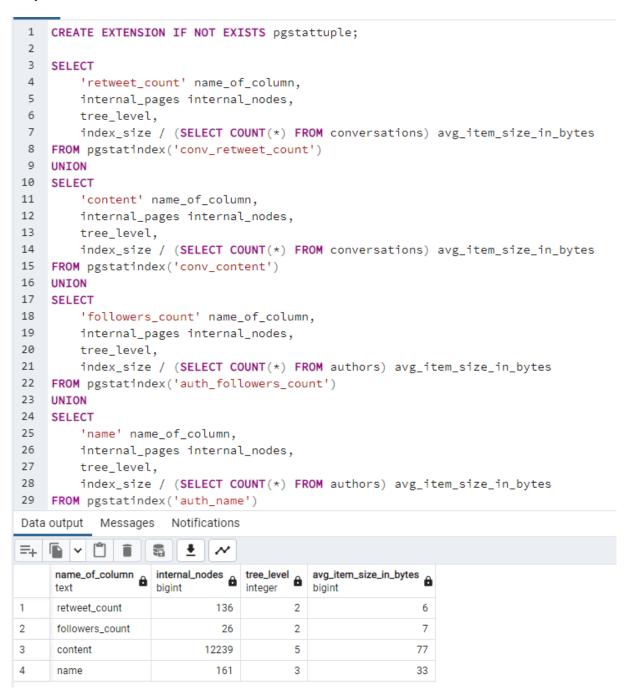
Dĺžka vytvárania indexu je ovplyvnená veľkosťou dát. Keďže stĺpec retweet_count pozostáva z jedného BIGINT, čo je 8B, tak vytváranie indexu nad ním bude oveľa rýchlejšie ako nad stĺpcom content, kde sú dlhé stringy dát, ktoré zberajú oveľa viacej miesta. Indexovanie je z veľkej časti zoraďovanie a menšie dáta sa zoraďujú ľahšie.

8. otázka

Otázka:

Porovnajte indexy pre retweet_count, content, followers_count, name,... v čom sa líšia pre nasledovné parametre: počet root nódov, level stromu, a priemerná veľkosť itemu. Vysvetlite.

Odpoveď:



Ako vidíme na obrázku, výsledky sa líšia v internal_nodes, tree_level a taktiež v priemernej veľkosti itemu. Veľkosť stromu (internal_nodes a tree_level) je

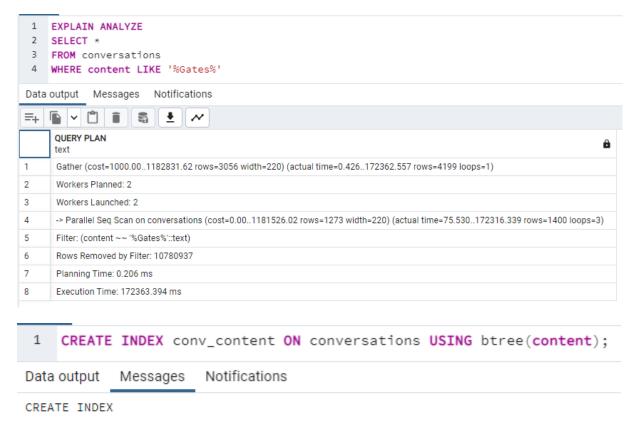
ovplyvnená najmä počtom dát a veľkosťou itemov. Veľkosť itemu je zas ovplyvnená formou dát, teda je samozrejmé, že dlhé reťazce budú mať väčšiu veľkosť ako čísla.

9. otázka

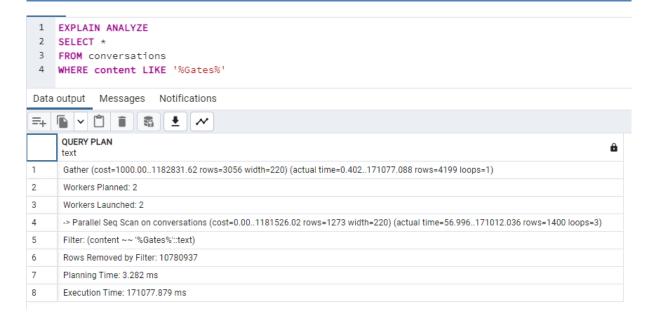
Otázka:

Vyhľadajte v conversations content meno "Gates" na ľubovoľnom mieste a porovnajte výsledok po tom, ako content naindexujete pomocou btree. V čom je rozdiel a prečo?

Odpoveď:



Query returned successfully in 12 min 41 secs.



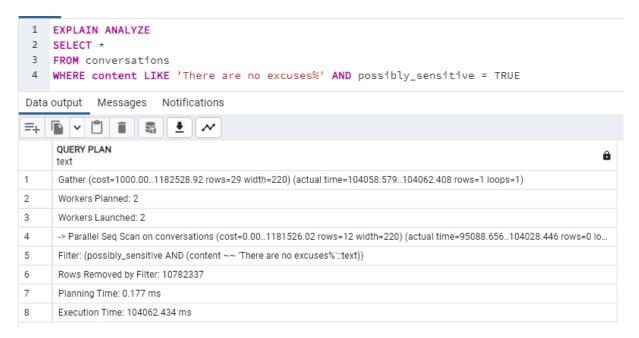
Jediný markantnejší rozdiel vidíme v Planning Time, čo môže byť spôsobené tým, že pri existencii indexu uvažuje plánovač nad jeho použitím. Pri oboch dopytoch bol použitý paralelný sekvenčný sken (s rovnakým počtom workerov). Index nevieme použiť kvôli tomu, že používame wildcard % na hľadanie v LIKE.

10. otázka

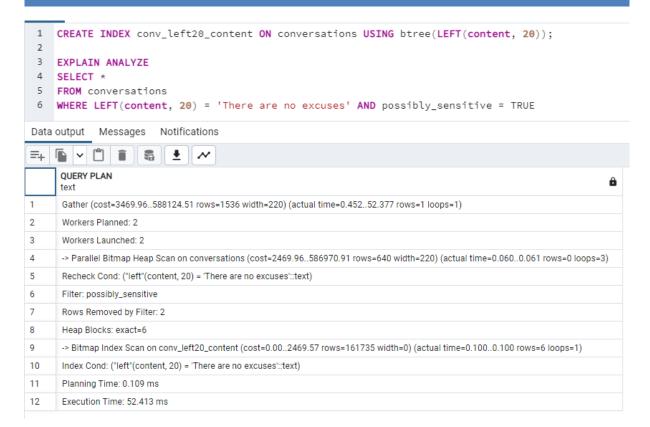
Otázka:

Vyhľadajte tweet, ktorý začína "There are no excuses" a zároveň je obsah potenciálne senzitívny (possibly_sensitive). Použil sa index? Prečo? Ako query zefektívniť?

Odpoveď:



Index sa nepoužil, keďže to nie je možné, ak nepoznáme **presný text** (to znamená ak máme v LIKE "wildcards" _ alebo %), ktorý chceme vyhľadať. Aby sme query zefektívnili, môžeme vytvoriť index na prvých 20 znakov stĺpca content. Tým pádom vieme spraviť vyhľadávanie podľa indexu, kde tweet začína s "There are no excuses", čo je presne 20 znakov.



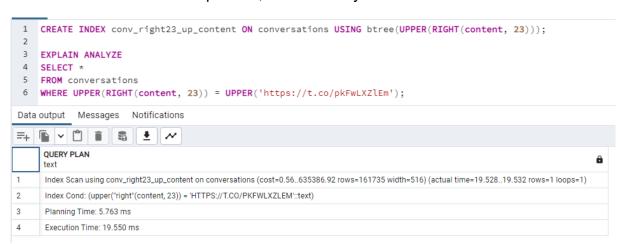
11. otázka

Otázka:

Vytvorte nový btree index, tak aby ste pomocou neho vedeli vyhľadať tweet, ktorý končí reťazcom "https://t.co/pkFwLXZIEm" kde nezáleží na tom ako to napíšete. Popíšte čo jednotlivé funkcie robia.

Odpoveď:

Aby sme vedeli použiť v query index, musíme použiť ten istý princíp ako v 10. úlohe, teda nepoužívať vyhľadávanie pomocou "wildcards" _ a %. To zabezpečíme vytvorením indexu pravej strany contentu, konkrétne 23 znakov, čo je rovnako veľa ako má náš reťazec, čo plánujeme vyhľadávať. Funkcia RIGHT() oreže reťazec na daný počet znakov. Túto istú funkciu potom musíme použiť aj pri vyhľadávaní, a to pri podmienke WHERE. Aby naše vyhľadávanie nebolo case sensitive, použijeme funkciu UPPER aj pre dáta z tabuľky content, aj pre dáta, čo hľadáme. Tým pádom nebude záležať na veľkosti písmen, keďže všetky zmeníme na veľké.

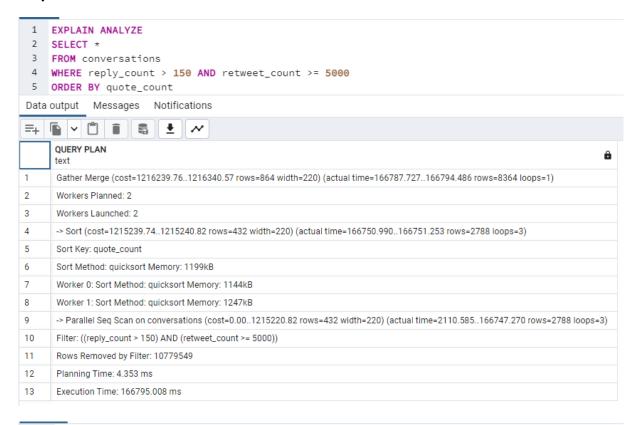


12. otázka

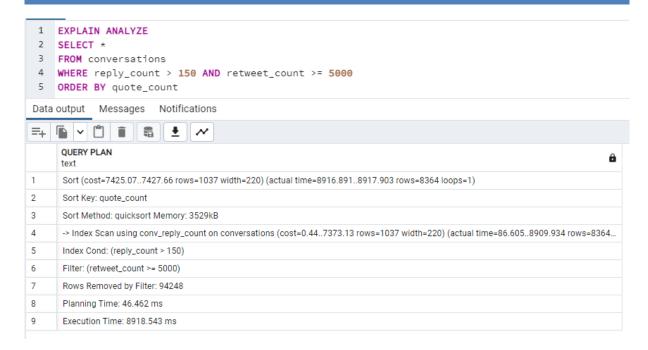
Otázka:

Nájdite conversations, ktoré majú reply_count väčší ako 150, retweet_count väčší rovný ako 5000 a výsledok zoraďte podľa quote_count. Následne spravte jednoduché indexy a popíšte ktoré má a ktoré nemá zmysel robiť a prečo. Popíšte a vysvetlite query plan, ktorý sa aplikuje v prípade použitia jednoduchých indexov.

Odpoveď:



```
1    CREATE INDEX conv_reply_count ON conversations USING btree(reply_count);
2    CREATE INDEX conv_retweet_count ON conversations USING btree(retweet_count);
3    CREATE INDEX conv_quote_count ON conversations USING btree(quote_count);
```



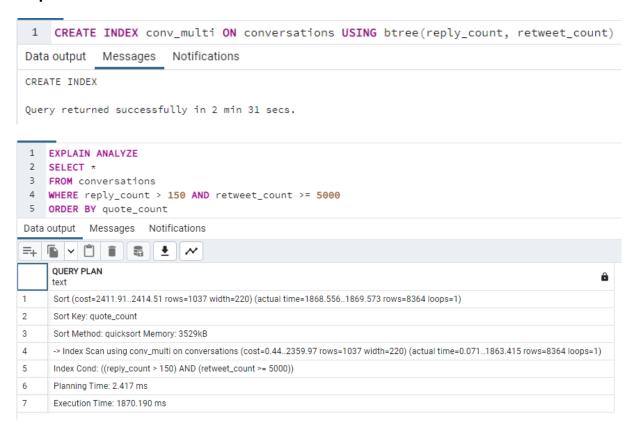
Z query plánu vidíme, že sa použil len index vytvorený nad stĺpcom reply_count. Je to z toho dôvodu, že pri využívaní indexu nedáva zmysel pre databázu použiť 2 rôzne indexy (pretože ak zredukujeme množinu výsledkov prvou podmienkou, tak v ďalšej podmienke už nepracujeme s dátami, nad ktorými bol index vytvorený). Ak by sme chceli použiť index na obe podmienky, potrebovali by sme zložený index.

13. otázka

Otázka:

Na predošlú query spravte zložený index a porovnajte výsledok s tým, keď sú indexy separátne. Výsledok zdôvodnite. Popíšte použitý query plan. Aký je v nich rozdiel?

Odpoveď:



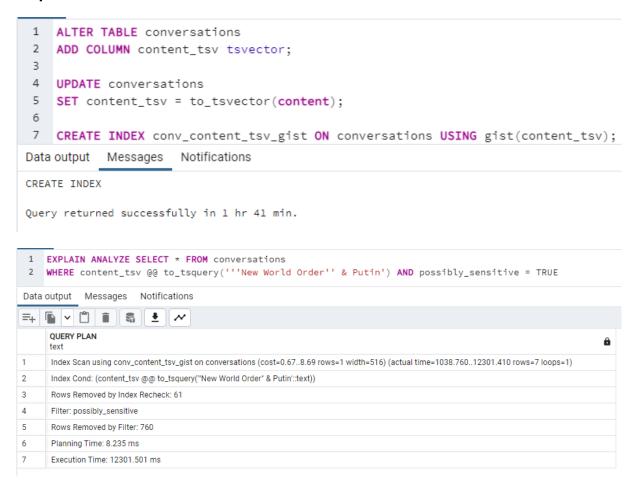
Z výsledku vidíme, že oproti separátnym indexom je zložený značne rýchlejší, a to z toho dôvodu, že pre obidve podmienky vo WHERE sa využije jeden a ten istý index (ako vidíme vo výstupe na riadku 5). Riešenie používajúce separátny index (v otázke 12) používa index len na prvú podmienku (riadok číslo 5 vo výstupe), zatiaľ čo druhá je riešená bez použitia indexu (riadok číslo 6-7).

14. otázka

Otázka:

Napíšte dotaz tak, aby sa v obsahu konverzácie našlo slovo "Putin" a zároveň spojenie "New World Order", kde slová idú po sebe a zároveň obsah je senzitívny. Vyhľadávanie má byť indexe. Popíšte použitý query plan pre GiST aj pre GIN. Ktorý je efektívnejší?

Odpoveď:





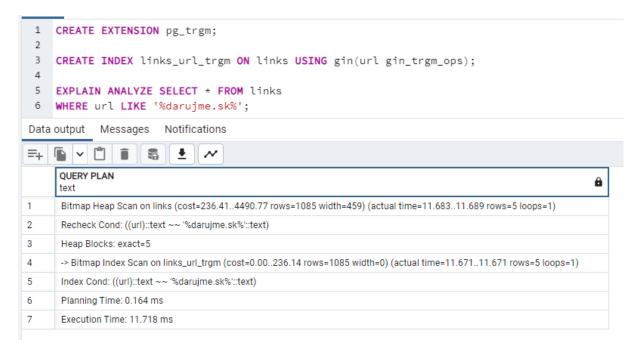
Z výsledkov vidíme, že pre GiST sa použil klasický index scan. Na druhej strane pri indexe GIN sa použil Bitmap Heap Scan (ktorý sme vysvetlili v otázke číslo 5) a musela sa aj "rechecknút" naša podmienka. Podľa časov ľahko vidíme, že v tomto prípade bolo použitie GIN indexu oveľa efektívnejšie.

15. otázka

Otázka:

Vytvorte vhodný index pre vyhľadávanie v links.url tak aby ste našli kampane z 'darujme.sk'. Ukážte dotaz a použitý query plan. Vysvetlite prečo sa použil tento index.

Odpoveď:



Použili sme index GIN využívajúci trigramy. Tsvector a tsquery neprichádzajú do úvahy, keďže sa hodia na vyhľadávanie slov a nie vyhľadávanie substringov (ako je napr. "darujme.sk" v strede URL adresy). Na vyhľadávanie substringov sú ideálne trigramy. Tento index sa potom využije pri použití klasického LIKE s wildcards (% alebo _).

16. otázka

Otázka:

Vytvorte query pre slová "Володимир" a "Президент" pomocou FTS (tsvector a tsquery) v angličtine v stĺpcoch conversations.content, authors.decription a authors.username, kde slová sa môžu nachádzať v prvom, druhom ALEBO treťom stĺpci. Teda vyhovujúci záznam je ak aspoň jeden stĺpec má "match". Výsledky zoradíte podľa retweet_count zostupne. Pre túto query vytvorte vhodné indexy tak, aby sa nepoužil ani raz sekvenčný scan (správna query dobehne rádovo v milisekundách, max sekundách na super starých PC). Zdôvodnite čo je problém s OR podmienkou a prečo AND je v poriadku pri joine.

Odpoveď:

Ak sme vytvárali tsvector v angličtine (tým pádom slová v azbuke neboli lematizované), mali sme problém, že hľadané slová boli v authors.description a v authors.username vyskloňované (napríklad mali 'a' na konci) a nenastala zhoda so slovami v tsquery. Preto sme sa rozhodli vyskúšať riešiť úlohu aj s lematizáciou, aj bez nej. Jedným z problémov, ktoré sme riešili, bol, ako priradiť konverzáciu k autorovi, ktorý má match v description (keďže jeden autor môže mať viacero konverzácií). Preto autorské zhody riešime samostatne a na konci ich pridáme pomocou UNION s tým, že chýbajúce údaje konverzácie vyplníme s NULLs. Zhody v conversations dopĺňame dátami z autorskej tabuľky pomocou JOIN. Ak by sme chceli použiť query aj na druhú tabuľku v rámci klauzuly JOIN, teda by sme použili OR, nepoužil by sa nám index a taktiež by sme kvôli prvej podmienke (author_id = id) stratili záznamy o používateľoch, ktorí nemajú žiaden záznam v konverzáciach.

S (ruskou) lematizáciou

```
-- with (russian) lemmatization--

ALTER TABLE conversations

ADD COLUMN content_tsv tsvector;

UPDATE conversations

SET content_tsv = to_tsvector('russian', content);

CREATE INDEX conv_content_tsv_gin ON conversations USING gin(content_tsv);
```

```
51 --select query (with lem)--
52 EXPLAIN ANALYZE
53 SELECT *
54 FROM conversations con
     FROM conversations con

JOIN authors aut ON con.author_id = aut.id AND con.content_tsv @@ to_tsquery('english', 'Володимир & Президент')
      SELECT NULL, *FROM authors
      WHERE multi_tsv @@ to_tsquery('english', 'Володимир & Президент')
 60 ORDER BY reply count DESC
 Data output Messages Notifications
QUERY PLAN
text
         Sort (cost=10262.94..10264.96 rows=810 width=1289) (actual time=8.210..8.259 rows=1060 loops=1)
      Sort Key: con.reply_count DESC

    → Unique (cost+10179.26.10223.81 rows=810 width+1289) (actual time+6.940..7.657 rows=1060 loops=1)
    → Sort (cost=10179.26.10181.28 rows=810 width=1289) (actual time+6.939..7.012 rows=1060 loops=1)

       Sort Key: con.id, con.author_id, con.content, con.possibly_sensitive, con.language, con.source, con.retweet_count, con.reply_count, con.like_count, con.quote_count, con.created_at, con.content_tsv, aut.id, aut.name, aut.username, aut.description, aut.f.
       -> Append (cost=58.70..10140.13 rows=810 width=1289) (actual time=0.623..5.735 rows=1060 loops=1)
        -> Nested Loop (cost=58.70..10079.96 rows=809 width=514) (actual time=0.622..5.636 rows=1058 loops=1)
       → Bitmap Heap Scan on conversations con (cost=58.27..3275.91 rows=809 width=252) (actual time=0.613..1.710 rows=1058 loops=1)
       Recheck Cond: (content_tsv @@ "володимир" & "президент"::tsquery)
      Heap Blocks: exact=1050
13
       -> Bitmap Index Scan on conv. content. tsv. gin (cost=0.00..58.07 rows=809 width=0) (actual time=0.498..0.498 rows=1058 loops=1)
      Index Cond: (content_tsv @@ "володимир" & "президент"::tsquery)
       Index Cond: (id = con.author_id)
        -> Bitmap Heap Scan on authors (cost=44.01..48.02 rows=1 width=431) (actual time=0.031..0.033 rows=2 loops=1)
       Recheck Cond: (multi_tsv @@ "володимир" & "президент"::tsquery)
        -> Bitmap Index Scan on auth_multi_tsv_gin (cost=0.00..44.01 rows=1 width=0) (actual time=0.027..0.027 rows=2 loops=1)
       Index Cond: (multi_tsv @@ "володимир" & "президент"::tsquery)
        Execution Time: 8.679 ms
Total rows: 23 of 23   Query complete 00:00:00.114
                                                                                                                                                                                                                                       Ln 59, Col 16
```

Bez lematizácie

```
1 -- without lemmatization--
   ALTER TABLE conversations
   ADD COLUMN content_tsv_en tsvector;
5 UPDATE conversations
   SET content_tsv_en = to_tsvector('english', content);
8 CREATE INDEX conv_content_tsv_en_gin ON conversations USING gin(content_tsv_en);
9
10
11
   ALTER TABLE authors
12 ADD COLUMN multi_tsv_en tsvector;
13
14 UPDATE authors
   SET multi_tsv_en = to_tsvector('english', description || ' ' || username);
15
16
   CREATE INDEX auth_multi_tsv_en_gin ON authors USING gin(multi_tsv_en);
```

(Bohužiaľ, index sa už nestihol spraviť pre riešenie bez lematizácie, a to z dôvodu pomalšieho počítaču a časového sklzu (56)