Fakulta informatiky a informačných technológií STU v Bratislave

Zadanie 2 – vyhľadávanie a indexovanie

29.9.2022

Matej Delinčák

Github classroom: https://github.com/FIIT-DBS/zadanie-pdt-mateju25

1. Vyhľadajte v authors username s presnou hodnotou 'mfa_russia' a analyzujte dany select. Aku metodu vam vybral planovač a prečo - odovodnite prečo sa rozhodol tak ako sa rozhodol?

Plánovač vybral paralelný sekvenčný sken, kedže pre daný stĺpec neexistuje žiaden index. Sekvenčný sken je vlastne to, že musíme prejsť riadok po riadku celú tabuľku. Paralelný preto, lebo podmienka dostatočne obmedzuje výsledok. Napríklad ak by sme nehľadali presnú hodnotu ale LIKE, tak čas by stúpol o nejakých 50 percent.

Výsledok:

	username character varying (255)
1	mfa_russia

Explain analyze:

1	Gather (cost=1000.00..147133.68 rows=587 width=11) (actual time=1197.776..1208.230 rows=1 loops=1)
2	[...] Workers Planned: 2
3	[...] Workers Launched: 2
4	[...] -> Parallel Seq Scan on authors (cost=0.00..146074.98 rows=245 width=11) (actual time=780.001..1135.485 rows=0 loops=3)
5	[...] Filter: ((username)::text ~~ 'mfa_russia'::text)
6	[...] Rows Removed by Filter: 1965058
7	Planning Time: 0.113 ms
8	Execution Time: 1208.254 ms

2. Koľko workerov pracovalo na danom selecte a na čo slúžia? Zdvihnite počet workerov a povedzte ako to ovplyvňuje čas. Je tam nejaky strop? Ak ano, prečo? Od čoho to zavisi (napište a popište všetky parametre)?

Na selecte pracovali dva worker-y. Išlo o paralelný sekvenčný sken a teda pri skenovaní budú bloky tabuľky rozdelené medzi worker-y. Bloky sa rozdávajú jeden po druhom, takže prístup k tabuľke zostáva sekvenčný.

Postupne som dvíhal strop počtu workerov a najlepší výsledok bol pri 8, ale reálne sa použilo len 7. Áno je tam strop. Jednak sa to týka konfigurácie samotného PostgreSQL:

- *max_worker_processes*: maximálny počet procesov na pozadí, ktoré systém podporovať
- *max_parallel_workers_per_gather*: počet workerov, ktoré môžu byť použíté pri skene
- *min_parallel_table_scan_size*: minimálna veľkosť tabuľky, aby sa použil sekvenčný sken
- *parallel_setup_cost*: parameter pre plánovač, koľko bude stáť priradenie workera
- *parallel_tuple_cost*: parameter pre plánovač, koľko bude stáť prenos dát medzi workerom

A teda strop pre výkon tam je vtedy, keď cena nastavovania workerov, prenos dát medzi nimi (main worker) a spájanie výsledkov odhadom trvá dlhšie ako obyčajný sekvenčný sken. Preto keď máme podmienku, ktorá nám odfiltruje veľký počet riadkov, plánovač sa rozhodne pre paralelizmus.

Systémový strop na počet workerov je podľa dokumentácie 1024.

Výsledok:

V prechádzajúcej odpovedi.

Explain analyze:

1	Gather (cost=1000.00..125692.31 rows=1 width=11) (actual time=1071.186..1083.651 rows=1 loops=1)
2	[...] Workers Planned: 8
3	[...] Workers Launched: 7
4	[...] -> Parallel Seq Scan on authors (cost=0.00..124692.21 rows=1 width=11) (actual time=817.454..898.260 rows=0 loops=8)
5	[...] Filter: ((username)::text = 'mfa_russia'::text)
6	[...] Rows Removed by Filter: 736897
7	Planning Time: 0.145 ms
8	Execution Time: 1083.679 ms

3. Vytvorte btree index nad username a pozrite ako sa zmenil čas a porovnajte vystup oproti požiadavke bez indexu. Potrebuje planovač v tejto požiadavke viac workerov? Čo ovplyvnilo zasadnu zmenu času?

Plánovač síce potreboval na plánovanie viac času, ale samotný dopyt trval radovo menej. Oproti paralelnému sekvenčnému skenu sa použil index only scan, ktorý síce nie je paralelný, ale za to pracuje na omnoho lepšej časovej zložitosti a to O(log N). Kedže nešlo o paralelný sken, na tomto dopyte pracoval len 1 worker. Paralelný sken by sa pravdepodobne použil vtedy, ak by sme nehľadali len jeden konkrétny záznam (pravdepodobne preto, lebo samotné manažovanie workerov by prebilo čas hľadania)

Výsledok:

V odpovedi 1.

Explain analyze:

1	Index Only Scan using idx_username on authors (cost=0.43..8.45 rows=1 width=11) (actual time=6.106..6.110 rows=1 loops=1)
2	[...] Index Cond: (username = 'mfa_russia'::text)
3	[...] Heap Fetches: 1
4	Planning Time: 2.024 ms
5	Execution Time: 6.129 ms

4. Vyberte používateľov, ktory maju followers_count vačši, rovny ako 100 a zároveň menši, rovny 200. Potom zmeňte rozsah na vačši, rovny ako 100 a zároveň menši, rovny 120. Je tam rozdiel, ak ano prečo?

Ak sme použili väčšiu podmienku, tak plánovač vybral sekvenčný sken. Ak sme podmienku obmedzili na menší rozsah, plánovač sa už rozhodol pre paralelný sken. Ako som písal v odpovedi pre otázku 2, je to preto, lebo keď máme menej záznamov, ktoré vyhovujú podmienke, ich cena na výsledné spojenie a manažovanie workerov je menšia ako obyčajný sekvenčný sken.

Výsledok:

<= 200

1	1391403261748338695	SjWealth	196
2	1204043515505729537	qptt14	144
3	369490089	ZehraAyd	185
4	1107461437918633984	_chavafloresr	185
5	1142639009962721280	cheche28061	194

<= 120

1	1420186765705502720	BFJ4KmSbIffIqLE	112
2	1167033716	Anichu_u	102
3	816767174	aeAble05	120
4	55795174	mrafieq	105
5	1493830171	hzorlu83	102

Explain analyze:

<= 200

1	Seq Scan on authors (cost=0.00..203908.64 rows=760822 width=146) (actual time=0.032..1684.533 rows=760088 loops=1)
2	[...] Filter: ((followers_count >= 100) AND (followers_count <= 200))
3	[...] Rows Removed by Filter: 5135088
4	Planning Time: 0.190 ms
5	Execution Time: 1707.973 ms

<= 120

1	Gather (cost=1000.00..156444.88 rows=178590 width=147) (actual time=0.530..767.109 rows=199937 loops=1)
2	[...] Workers Planned: 4
3	[...] Workers Launched: 4
4	[...] -> Parallel Seq Scan on authors (cost=0.00..137585.88 rows=44648 width=147) (actual time=0.050..560.694 rows=39987 loo...
5	[...] Filter: ((followers_count >= 100) AND (followers_count <= 120))
6	[...] Rows Removed by Filter: 1139048
7	Planning Time: 0.114 ms
8	Execution Time: 787.774 ms

5. Vytvorte index nad 4 ulohou a v oboch podmienkach popište pracu s indexom. Čo je to Bitmap Index Scan a prečo je tam Bitmap Heap Scan? Prečo je tam recheck condition? Použil sa vždy index?

Index sa použil len v prípade, keď bol rozsah v podmienke väčší. Ale index napriek tomu zrýchlil dopyt pri podmienke <= 200. Oproti indexu na username, kde boli všetky hodnoty unique, pre tento stĺpec je len asi 100k záznamov unique. Teda máme veľké overflow stránky. Bitmap Index Scan skenuje index podľa podmienok takmer rovnakým spôsobom ako pri bežnom Index Scan. Ale namiesto vrátenia TID (pozostávajúceho z čísla stránky a posunu v ňom) dát z heapu, pridá TID do bitovej mapy. Táto bitmapa obsahuje hash všetkých stránok a každý záznam stránky obsahuje pole všetkých posunov v rámci tejto stránky. Potom Bitmap Heap Scan prečíta bitovú mapu, aby získal dáta z heapu zodpovedajúce uloženému číslu stránky a posunu. Recheck condition je vtedy, ak bitová mapa, ktorá má 1 bit pre riadok tabuľky, je väčšia ako *work_mem.* Čiže sa nezmestí do pamäte a PostreSQL priradí 1 bit len pre stránku nie riadok. Následne riadky z takýchto stránok musia byť znova podrobené podmienke.

Výsledok:

<= 200

1	2739495449	orionqqq	134
2	2883591468	SneakySnake26	105
3	1277857064874725377	cybxrlawo	127
4	17663396	evanthegreat	171
5	1431138794078707712	Kind_Of_Magik_	162

1	2883591468	SneakySnake26	105
2	1092989510370107392	_maaraandaa	118
3	1076682246327693312	natuno0725	110
4	846246158716420098	ChweBonon_	102
5	980806431367507973	masturdatingo	117

<= 120

Explain analyze:

<= 200

1	Bitmap Heap Scan on authors (cost=10530.86..192586.22 rows=760822 width=146) (actual time=97.628..1058.062 rows=76008...
2	[...] Recheck Cond: ((followers_count >= 100) AND (followers_count <= 200))
3	[...] Rows Removed by Index Recheck: 2928778
4	[...] Heap Blocks: exact=48745 lossy=66190
5	[...] -> Bitmap Index Scan on idx_followers_count (cost=0.00..10340.65 rows=760822 width=0) (actual time=82.332..82.332 rows...
6	[...] Index Cond: ((followers_count >= 100) AND (followers_count <= 200))
7	Planning Time: 0.164 ms
8	Execution Time: 1088.203 ms

<= 120

1	Gather (cost=1000.00..157526.41 rows=189385 width=146) (actual time=0.669..762.302 rows=199937 loops=1)
2	[...] Workers Planned: 4
3	[...] Workers Launched: 4
4	[...] -> Parallel Seq Scan on authors (cost=0.00..137587.91 rows=47346 width=146) (actual time=0.049..590.806 rows=39987 loo...
5	[...] Filter: ((followers_count >= 100) AND (followers_count <= 120))
6	[...] Rows Removed by Filter: 1139048
7	Planning Time: 0.121 ms
8	Execution Time: 776.342 ms

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

Po vymazaní indexov bol insert rýchlejší. Je to preto, lebo po prvé index nemá podmienku a teda priamo nebenefituje nijako z indexu. Po druhé preto, lebo index sa musí zachovať vyvážený (BTREE). Cena udržiavania indexu je nejaká a teda bez indexov nám to odpadáva. A teda čím viac indexov máme pre danú tabuľku, tým dlhšie bude insert trvať.

Výsledok:

Záznam sa vložil.

```
INSERT INTO public.authors(
    username, name, description, followers_count, following_count, listed_count, tweet_count)
    VALUES ('', '', '', 0,0,0,0);
```

Explain analyze:

S indexami

1	Insert on authors (cost=0.00..0.01 rows=0 width=0) (actual time=0.205..0.205 rows=0 loops=1)
2	[...] -> Result (cost=0.00..0.01 rows=1 width=1088) (actual time=0.010..0.011 rows=1 loops=1)
3	Planning Time: 0.040 ms
4	Execution Time: 0.229 ms

Bez indexov

1	Insert on authors (cost=0.00..0.01 rows=0 width=0) (actual time=0.061..0.061 rows=0 loops=1)
2	[...] -> Result (cost=0.00..0.01 rows=1 width=1088) (actual time=0.001..0.001 rows=1 loops=1)
3	Planning Time: 0.027 ms
4	Execution Time: 0.073 ms

7. Vytvorte btree index nad conversations pre retweet_count a pre content. Porovnajte ich dĺžku vytvarania. Prečo je tu taky rozdiel? Čim je ovplyvnena dĺžka vytvarania indexu a prečo?

V krátkosti retweet_count je číselná hodnota a s ňou sa pracuje omnoho rýchlejšie ako s textovou hodnotu content. Pri stringoch treba porovnávať znak po znaku, číslo stačí porovnať raz. Čo sa týka pamäte, aj tu má číslo výhodu, kedže integer zaberá 4B a stĺpec content typu text môže obsahovať až 65k znakov (viac ako 65kB). Túto skutočnosť môžeme vidieť aj na výsledku, ktorý som dostal nižšie. Rozdiel je +- 10-násobný.

Výsledok:

retweet_count

```
CREATE INDEX

Query returned successfully in 46 secs 738 msec.
```

content

```
CREATE INDEX

Query returned successfully in 9 min 40 secs.
```

Explain analyze:

Explain analyze sa nedá použiť na create index.

8. Porovnajte indexy pre retweet_count, content, followers_count, name,... v čom sa líšia pre nasledovne parametre: počet root nodov, level stromu, a priemerna veľkosť itemu. Vysvetlite.

Asi ste sa pytali na internal nodes a nie na root node. Pre kazdy index je root node len jeden.

	Internal nodes	Level of tree	Average item size
retweet_count (4B)	136	2	23
content (65kB)	12239	5	188
followers_count (4B)	26	2	22
name (255B)	161	3	32
username (255B)	106	2	23
description (65kB)	2022	4	226

Veľkosť jednotlivých parametrov závisí od veľkosti stránky, ktoré postgres používa (8kB), záleží od počtu záznamov, od veľkosti kľúča podľa, ktorého robíme index. To znamená, že koľko kľúčov sa zmestí do jednej stránky a samozrejme aj od unikátnosti dát.

Na zobrazenie hĺbky stromu a počtu rood nodes som použil príkaz, kde som menil názvy indexov.

```
SELECT * FROM pgstatindex('idx_name'::regclass);
```

A pre priemernú veľkosť itemu som použil, kde id pagu som dával root node daného indexu:

```
SELECT * FROM bt_page_stats('idx_name', 22068);
```

Výsledok:

Nezobrazím výsledok každej query, len tejto jednej vybratej pre prvý príkaz.

```
31 select * from pgstatindex('idx_retweet_count'::regclass);
32
```

Data Output | Explain | Messages | Notifications

	version integer	tree_level integer	index_size bigint	root_block_no bigint	internal_pages bigint	leaf_pages bigint	empty_pages bigint	deleted_pages bigint	avg_leaf_density double precision	leaf_fragmentation double precision
1	4	2	225804288	209	136	27427	0	0	89.49	0

Pre druhý

```
34 SELECT * FROM bt_page_stats('idx_retweet_count', 209);
35
```

Data Output | Explain | Messages | Notifications

	blkno bigint	type "char" (1)	live_items integer	dead_items integer	avg_item_size integer	page_size integer	free_size integer	btpo_prev bigint	btpo_next bigint	btpo_level bigint	btpo_flags integer
1	209	r	135	0	23	8192	4408	0	0	2	2

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

V oboch prípadoch aj keď nebol a aj keď bol index sa použil paralel seq scan. Je to preto, lebo BTREE index nám nepomôže pri hľadaní výrazu, ktorý sa môže nachádzať hocikde v texte. Význam indexu by bol vtedy, ak by sme hľadali práve taký text, kde je vyslovene len „Gates“ (WHERE content = ‘Gates’). Ak som zakázal použitie seq scanu, tak to reálne pomocou použitia indexu trvalo omnoho dlhšie. BTREE index totiž nezohľadňuje kontext.

Výsledok:

	content text
1	RT @BillinderGates: Aajao milne 😎😂
2	RT @BillinderGates: Aajao milne 😎😂
3	RT @BillinderGates: Aajao milne 😎😂
4	@tim_cook @elonmusk @BillGates @JeffBezos STOP BUSINESS WITH RUSSIA NOW !!! #totalembargo #totalsanctionrussia #Russia #Ukraine #UkraineRussia #Tesla @Tesla @MSFT_BusinessDE @thomashelmer5 @ntvde @FinancialTimes @cnnbrk @dw_busin...
5	RT @4n4lisis: Interesting visit of Bill Gates to Pakistan.

Explain analyze:

Bez indexu aj s indexom to bude to isté.

1	Gather (cost=1000.00..1109050.23 rows=3057 width=158) (actual time=0.767..18741.584 rows=4199 loops=1)
2	[...] Workers Planned: 5
3	[...] Workers Launched: 5
4	[...] -> Parallel Seq Scan on conversations (cost=0.00..1107744.53 rows=611 width=158) (actual time=43.632..18616.893 rows=7...
5	[...] Filter: (content ~~ '%Gates%'::text)
6	[...] Rows Removed by Filter: 5390469
7	Planning Time: 0.295 ms
8	Execution Time: 18742.404 ms

10. Vyhľadajte tweet, ktory začina “There are no excuses” a zaroveň je obsah potencialne senzitivny (possibly_sensitive). Použil sa index? Prečo? Ako query zefektivniť?

Pri tejto otázke som mal problém s tým, že index to malo použiť (oproti otázke 9), pretože hľadáme začiatok kľúča. Ale dočítal som sa v dokumentácií https://www.postgresql.org/docs/8.4/indexes-types.html , že ak nemám C locale, tak indexy nepoznajú pattern-matching queries (čiže príkaz LIKE ‘%abcd%’). Preto keď som už použil takýto command na vytvorenie indexu, tak mi to ten index už použilo:

```
CREATE INDEX idx_content ON conversations(content
                text_pattern_ops);
```

Na zefektívnenie som neprišiel na nič, podľa mňa je už dosť rýchla. Teda ak ste nemysleli náhodou to, že predtým sa index nepoužil, vykonal sa len seq scan a keď som spravil už správny index tak ten sa použil.

Výsledok:

Data Output | Explain | Messages | Notifications

	id [PK] bigint	content text	possibly_sensitive boolean
1	1507587876355149825	There are no excuses, zero, non, zilch, nada.	true

Explain analyze:

Pred zefektivnenim:

1	Index Scan using idx_content_new on conversations (cost=0.81..8.83 rows=34 width=158) (actual time=0.059..0.062 rows=1 loop...
2	[...] Index Cond: ((content ~>=~ 'There are no excuses'::text) AND (content ~<~ 'There are no excuset'::text))
3	[...] Filter: ((possibly_sensitive IS TRUE) AND (content ~~ 'There are no excuses%'::text))
4	[...] Rows Removed by Filter: 5
5	Planning Time: 0.659 ms
6	Execution Time: 0.093 ms

11. Vytvorte novy btree index, tak aby ste pomocou neho vedeli vyhľadať tweet, ktory končí reťazcom „https://t.co/pkFwLXZlEm“ kde nezaleži na tom ako to napišete. Popište čo jednotlive funkcie robia.

Najprv som si vytvoril špeciálny index na suffixy stĺpca štýlom:

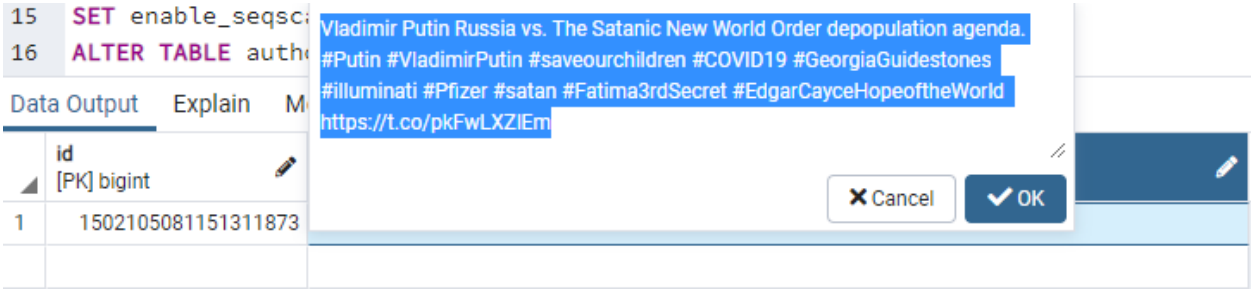
```
CREATE INDEX idx_content_reverse ON
conversations(reverse(lower(content)) text_pattern_ops);
```

A keď som chcel vyhľadať tak som musel query upraviť nasledovne:

```
EXPLAIN ANALYZE SELECT * FROM conversations WHERE
        reverse(lower(content)) like
    reverse(lower('%https://t.co/pkFwLXZlEm'))
```

Text teda môžeme napísať akokoľvek. V tomto indexe najskôr premením všetky znaky na malé písmena a potom obrátim reťazec. Pri hľadaní treba urobiť ten istý postup aj na hodnotu hľadania aj na stĺpec. A vtedy to použije index ako môžeme vidieť dole. Plánovač použil Index Scan na tomto opačnom indexe.

Výsledok:



Explain analyze:

1	Index Scan using idx_content_reverse on conversations (cost=0.81..610308.53 rows=161735 width=220) (actual time=0.030..0.030 rows=1 loops=1)
2	[...] Index Cond: ((reverse(lower(content)) ~>=~ 'melzxlwfkp/oc.t//:sptth'::text) AND (reverse(lower(content)) ~<~ 'melzxlwfkp/oc.t//:sptti'::text))
3	[...] Filter: (reverse(lower(content)) ~~ 'melzxlwfkp/oc.t//:sptth%'::text)
4	Planning Time: 0.157 ms
5	Execution Time: 0.043 ms

12. Najdite conversations, ktore maju reply_count vačši ako 150, retweet_count vačši rovny ako 5000 a vysledok zoraďte podľa quote_count. Nasledne spravte jednoduche indexy a popište ktore ma a ktore nema zmysel robiť a prečo. Popište a vysvetlite query plan, ktory sa aplikuje v pripade použitia jednoduchych indexov.

Pri tejto otázke je dôležité si uvedomiť, že najlepšie je to aby sa čo najrýchlejšie odfiltrovali záznamy, ktoré nepotrebujeme. Po prezretí dát som našiel, že väčšina reply_count je null. Teda ak použijem index na tento stĺpec, táto podmienka je silno selektívna a bude to rýchle. Tým pádom nám netreba robiť index na retweet_count, lebo podmienka je málo selektívna a sa použije radšej sekvenčný sken. Keď som skúšal len index pre quote_count, tak tam rozdiel nebol, pretože stále musíme prejsť všetky záznamy a pozrieť, či platí podmienka. Zároveň bez správneho indexu sa použil paralelný sekvenčný sken, kde dáta vo workery sa zoradili a následne cez GatherMerge spojijli. Stačí nám teda index len na reply_count.

Výsledok:

	id [PK] bigint	language character varying (3)	content text	source text	retweet_count integer	reply_count integer	like_count integer	quote_count integer	po bo
1	1497786759710752768	en	$50 | 700K • 5 Hours	Twitter for Android	5397	2277	1163	0	fal
2	1498836083416584195	en	$50 | 700.000 IDR • End 8 hour	Twitter for iPhone	5632	1848	4723	1	fal
3	1504290852151390212	en	$100 • 1,4 JT | 12 HOURS	Twitter for iPad	10046	3721	2257	1	fal
4	1502965141998272513	en	$100 • 1,4 JT | 12 HOURS	Twitter for iPad	6561	3111	1587	1	fal
5	1498156335581573120	en	$50 | 700K • 4 Hours	Twitter for Android	5781	2373	1254	1	fal
6	1448484238904866560	en	What an asshole. #Putin knows #angelamerkel is afraid of dogs yet brings his anyway. Next time s	Twitter for Android	6621	345	2065	1	fal

Explain analyze:

Pomocou indexu sa rýchlo našli všetky riadky s reply_count > 100, potom sa vyfiltrovali pod+la druhej podmienky a nakoniec zoradili.

1	Sort (cost=77188.66..77219.19 rows=12212 width=220) (actual time=159.544..160.776 rows=8559 loops=1)
2	[...] Sort Key: quote_count
3	[...] Sort Method: quicksort Memory: 3644kB
4	[...] -> Index Scan using idx_reply_count on conversations (cost=0.44..76359.71 rows=12212 width=220) (actual time=0.188..155....
5	[...] Index Cond: (reply_count > 100)
6	[...] Filter: (retweet_count >= 5000)
7	[...] Rows Removed by Filter: 131043
8	Planning Time: 1.086 ms
9	Execution Time: 161.581 ms

13. Na predošlu query spravte zloženy index a porovnajte vysledok s tym, keďy je su indexy separatne. Vysledok zdovodnite. Popište použity query plan. Aky je v nich rozdiel?

Výsledok som dostal rýchlejšie ako v predošlej úlohe, pretože teraz som vytvoril index pomcou dvoch kľúčov, pre ktoré mám podmienky v query:

```
CREATE INDEX idx_conv_multiple ON conversations(reply_count, retweet_count);
```

Ako prvý do zloženého indexu som vložil reply_count stĺpec, pretože viacej riadkov sa vyfiltruje a ako druhé som použil retweet_count. Každý riadok na základe prvého stlpca v zloženom indexe ma ešte pointer do druhého indexu, kde sú hodnoty zoradené podľa druhého stĺpca.

V query pláne je vidno, že z indexu sa vyberajú také riadky, ktoré potrebujeme a už ich nemusíme ďalej filtrovať.

Výsledok:

	id [PK] bigint	language character varying (3)	content text	source text	retweet_count integer	reply_count integer	like_count integer	quote_count integer	pos bool
1	1497786759710752768	en	$50 | 700K • 5 Hours	Twitter for Android	5397	2277	1163	0	fals
2	1502965141998272513	en	$100 • 1,4 JT | 12 HOURS	Twitter for iPad	6561	3111	1587	1	fals
3	1498156335581573120	en	$50 | 700K • 4 Hours	Twitter for Android	5781	2373	1254	1	fals
4	444848423804866560	en	What an asshole. #Putin knows #angelamerkel is afraid of dogs yet brings his anyway. Next time s...	Twitter for Android	6621	345	2965	1	fals
5	1498836083416584195	en	$50 | 700.000 IDR • End 8 hour	Twitter for iPhone	5632	1848	4723	1	fals
6	1504290852151390212	en	$100 • 1,4 JT | 12 HOURS	Twitter for iPad	10046	3721	2257	1	fals

Explain analyze:

1	Sort (cost=28044.56..28075.09 rows=12212 width=220) (actual time=15.144..16.145 rows=8559 loops=1)
2	[...] Sort Key: quote_count
3	[...] Sort Method: quicksort Memory: 3644kB
4	[...] -> Index Scan using idx_conv_multiple on conversations (cost=0.44..27215.61 rows=12212 width=220) (actual time=0.032..1...
5	[...] Index Cond: ((reply_count > 100) AND (retweet_count >= 5000))
6	Planning Time: 0.163 ms
7	Execution Time: 16.624 ms

Skúšal som aj do indexu dať quote_count, ale tento index bol pomalší aj keď sa v query pláne nepoužil sort:

```
CREATE INDEX idx_conv_multiple_quote_count ON conversations(quote_count, reply_count, retweet_count);
```

	QUERY PLAN text
1	Index Scan using idx_conv_multiple_quote_count on conversations (cost=0.56..463390.60 rows=12212 width=220) (actual time=...
2	[...] Index Cond: ((reply_count > 100) AND (retweet_count >= 5000))
3	Planning Time: 0.198 ms
4	Execution Time: 57.162 ms

14. Napište dotaz tak, aby sa v obsahu konverzacie našlo slovo „Putin“ a zároveň spojenie „New World Order“, kde slova idu po sebe a zároveň obsah je senzitivny. Vyhľadavanie ma byť indexe. Popište použity query plan pre GiST aj pre GIN. Ktory je efektivnejši?

```
SELECT id, content FROM conversations WHERE content like
'%Putin%New World Order%' and possibly_sensitive = true
```

Vyhľadávanie v gin indexe bolo omnoho rýchlejšie ako v giste. V giste to trvalo príšerne dlho. V dokumentácií som sa dočítal, že to kvôli tomu, lebo gist index pracuje najlepšie ak máme menej ako 100k unique záznamov. Pre obe query plány sa najskôr použil daný index pre nájdenie stránok, potom sa odstránili také, ktoré nespĺňali podmienku a nakoniec sa recheckol condition na jednotlivé riadky. Prečo sa spravil recheck mám zodpovedané už v predým otázke.

Výsledok:

	id [PK] bigint	content text
1	1497653608028028931	#Putin and the New World Order from Damascus to Kiev
2	1498019842183544838	RT @veteranstoday: Putin: New World Order Worships Satan
3	1498483618804649990	#Putin Has Banned #Rothschild And His New World Order Banking Cartel Family From Entering #R...
4	1501745570045743104	#Putin is gonna Destroy The New World Order! ʀᴜ https://t.co/Vvha9YIwX4
5	1498341252927946758	Would Putin take responsibility for this? 6 year old Ukrainian girl killed during Russian invasion. Sta...

Explain analyze:

Gin:

1	Bitmap Heap Scan on conversations (cost=374.93..12125.74 rows=34 width=166) (actual time=777.206..778.322 rows=5 loops=1)
2	[...] Recheck Cond: (content ~~ '%Putin%New World Order%'::text)
3	[...] Rows Removed by Index Recheck: 870
4	[...] Filter: possibly_sensitive
5	[...] Rows Removed by Filter: 221
6	[...] Heap Blocks: exact=1095
7	[...] -> Bitmap Index Scan on gin_conversation (cost=0.00..374.92 rows=3057 width=0) (actual time=776.996..776.996 rows=109...
8	[...] Index Cond: (content ~~ '%Putin%New World Order%'::text)
9	Planning Time: 0.302 ms
10	Execution Time: 778.386 ms

Gist:

1	Bitmap Heap Scan on conversations (cost=1183.48..12934.29 rows=34 width=166) (actual time=693778.232..693886.935 rows=...
2	[...] Recheck Cond: (content ~~ '%Putin%New World Order%'::text)
3	[...] Rows Removed by Index Recheck: 870
4	[...] Filter: possibly_sensitive
5	[...] Rows Removed by Filter: 221
6	[...] Heap Blocks: exact=1095
7	[...] -> Bitmap Index Scan on gist_conversation (cost=0.00..1183.47 rows=3057 width=0) (actual time=693773.682..693773.683 r...
8	[...] Index Cond: (content ~~ '%Putin%New World Order%'::text)
9	Planning Time: 12.589 ms
10	Execution Time: 693888.023 ms

15. Vytvorte vhodny index pre vyhľadavanie v links.url tak aby ste našli kampane z ‘darujme.sk’. Ukažte dotaz a použity query plan. Vysvetlite prečo sa použil tento index.

```
EXPLAIN ANALYZE SELECT * FROM links WHERE url like
'%darujme.sk%'
```

Vytvoril som aj GIN aj GiST index, aby som svoje tvrdenie potvrdil. Ako píšem v otázke predtým, GiST je vhodný na dáta do 100k riadkov. Síce je na update rýchlejší ako GIN, ale to v tejto úlohe nehrá žiadnu rolu. A teda ako môžeme vidieť dole na query pláne, vybral som GIN, lebo bol rýchlejší. Zaujímavé je to, že ak mal plánovač oba indexy k dispozícií vybral ten pomalší GiST a nedočítal som sa nikde že prečo. Gin index sa použil preto, lebo iný som nedal k dispozicií.

Výsledok:

	id [PK] bigint	title text	description text	url character varying (2048)
1	1590852	Pomoc Ukrajine	To, čoho sme sa mesiace obávali, sa stalo skutočnosťou. Ruská federácia napadla Ukrajinu a konflikt, ktorý sa vymedzoval na separatistickú oblasť Donbasu, zasiahol do života miliónov ďalších ľudí. V	https://charita.darujme.sk/ukrajina/
2	4304874	[null]	[null]	https://clovekvohrozeni.darujme.sk/pomoc-uk
3	9176936	Pomáhame Ukrajine	Prioritou Červeného kríža je zmierňovať utrpenie ľudí. Pracovníci a dobrovoľníci Slovenského Červeného kríža sú pripravení pomáhať na hraniciach s Ukrajinou – poskytovať prvú pomoc, asistovať v prípad	https://redcross.darujme.sk/pomahame-ukraji
4	10672875	[null]	[null]	https://redcross.darujme.sk/pomahame-ukraji
5	11036620	[null]	[null]	https://zvieraciombudsman.darujme.sk/anima

Explain analyze:

1	Bitmap Heap Scan on links (cost=224.41..4345.62 rows=1085 width=360) (actual time=8.477..8.482 rows=5 loops=1)
2	[...] Recheck Cond: ((url)::text ~~ '%darujme.sk%'::text)
3	[...] Heap Blocks: exact=5
4	[...] -> Bitmap Index Scan on gin_urls (cost=0.00..224.14 rows=1085 width=0) (actual time=8.469..8.469 rows=5 loops=1)
5	[...] Index Cond: ((url)::text ~~ '%darujme.sk%'::text)
6	Planning Time: 0.131 ms
7	Execution Time: 8.561 ms

16. Vytvorte query pre slova "Володимир" a "Президент" pomocou FTS (tsvector a tsquery) v angličtine v stĺpcoch conversations.content, authors.decription a authors.username, kde slova sa možu nachadzať v prvom, druhom ALEBO treťom stĺpci. Teda vyhovujuci zaznam je ak aspoň jeden stĺpec ma „match“. Vysledky zoradite podľa retweet_count zostupne. Pre tuto query vytvorte vhodne indexy tak, aby sa nepoužil ani raz sekvenčny scan (spravna query dobehne radovo v milisekundach, max sekundach na super starych PC). Zdovodnite čo je problem s OR podmienkou a prečo AND je v poriadku pri joine.

V query pláne sa využili dva gin indexy pre tabuľku conversation na stlpec content, nad ktorým som dal tsvector a authors, kde som joinol dva stlpce a premenil na tsvector. Taktiež jeden btree index nad stĺpcom author_id v tabuľke conversations. Problém s OR je ten, že nedokáže použiť index. To som vyriešil tým, že select som rozdelil na dva selecty a jednotlivé podmienky som pridal do JOIN podmienky. AND v joine podmienke nevadí, lebo tam vieme využiť index.

```
EXPLAIN ANALYZE SELECT c.id, c.content, a.username, a.description, retweet_count from conversations c INNER JOIN authors a ON a.id = c.author_id
AND to_tsquery('english','Володимир & Президент') @@ to_tsvector('english', coalesce(a.username,'')||' '||coalesce(a.description,''))
UNION
SELECT c.id, c.content, a.username, a.description, retweet_count from authors a INNER JOIN  conversations c ON a.id = c.author_id
AND to_tsquery('english','Володимир & Президент') @@ to_tsvector('english', c.content)
ORDER BY retweet_count DESC
```

Z nudy som sa ešte pohral a dospel aj k takejto query, ktorá je asi o polovičku rýchlejšia. Ale funguje len pre tento dopyt. Ide tam o to, že ja authorov joinujem len už z vyfiltrovaných conversations. Teda ak je tam nejaký author, ktorý ma conversation, ktoré nemám, tak budem mat null content. Ale inak pre tento konkrétny prípad mi to dáva totožné výsledky.

```
WITH
 filtered_auths as (select a.* from authors a where to_tsquery('english','Володимир & Президент')
                 @@ to_tsvector('english', coalesce(a.username,'')||' '||coalesce(a.description,''))),
 filtered_conv as (select c.* from conversations c where to_tsquery('english','Володимир & Президент') @@ to_tsvector('english', c.content))
SELECT * FROM (
    SELECT c.id, c.content, a.username, a.description, retweet_count FROM filtered_conv c LEFT JOIN authors a  ON c.author_id = a.id UNION
    SELECT c.id, c.content, a.username, a.description, retweet_count FROM filtered_auths a LEFT JOIN filtered_conv c  ON a.id = c.author_id
) as result ORDER BY retweet_count DESC;
```

Výsledok:

	id bigint	content text	username character varying (255)	description text	retweet_count integer
1	1503026814444310532	Президент України Володимир Зеленський відвідав у госпіталі п…	DefenceU	Official page of the Ministry of Defense of Ukraine UA	5848
2	1498546324232359936	В бою загинув льотчик-винищувач Олександр Оксанченко.	UkrArmyBlog	СЛАВА УКРАЇНІ ТА ЇЇ ВОЇНАМ! Неофіційна сторінка підтр…	2799
3	1499904884950421509	Головнокомандувачу ЗС України генерал-лейтенанту Валерію За…	DefenceU	Official page of the Ministry of Defense of Ukraine UA	1489
4	1498329101823791104	▌▌▌▌Президент України Володимир Зеленський підписав заявку…	verkhovna_rada	Офіційна сторінка Верховної Ради України| Official account of Verkho…	1109
5	1498599832386252801	▌▌Сьогодні, 1 березня, Президент України Володимир Зеленськи…	verkhovna_rada	Офіційна сторінка Верховної Ради України| Official account of Verkho…	649
6	1498439122796834816	Президент Володимир Зеленський підписав указ про присвоєнн…	ua_industrial	My name is Pavlo. Here the whole world reads about the war in Ukraine …	472
7	1497834303757197312	Президент України Володимир Зеленський звернувся до всіх гр…	UkrArmyBlog	СЛАВА УКРАЇНІ ТА ЇЇ ВОЇНАМ! Неофіційна сторінка підтр…	433
8	1497444480118575105	Президент у відповідь на пропозицію Байдена щодо евакуації:	ltpzdc	Тут атланти-терикони підпирають небокрай	423

Explain analyze:

1	Sort (cost=41016.31..41020.35 rows=1616 width=592) (actual time=124.018..124.242 rows=838 loops=1)
2	[...] Sort Key: c.retweet_count DESC
3	[...] Sort Method: quicksort Memory: 508kB
4	[...] -> HashAggregate (cost=40914.04..40930.20 rows=1616 width=592) (actual time=122.654..123.230 rows=838 loops=1)
5	[...] Group Key: c.id, c.content, a.username, a.description, c.retweet_count
6	[...] Batches: 1 Memory Usage: 585kB
7	[...] -> Append (cost=1045.58..40893.84 rows=1616 width=592) (actual time=114.982..121.202 rows=838 loops=1)
8	[...] -> Gather (cost=1045.58..30679.04 rows=807 width=277) (actual time=114.051..114.172 rows=0 loops=1)
9	[...] Workers Planned: 2
10	[...] Workers Launched: 2
11	[...] -> Nested Loop (cost=45.58..29598.34 rows=336 width=277) (actual time=0.076..0.077 rows=0 loops=3)
12	[...] -> Parallel Bitmap Heap Scan on authors a (cost=45.14..633.79 rows=61 width=115) (actual time=0.075..0.076 rows=0 loops=3)
13	[...] Recheck Cond: ("володимир" & "президент"::tsquery @@ to_tsvector('english'::regconfig, (((COALESCE(username, "::character varying))::text || ' '::text) || COALESCE(description, "::text))))
14	[...] -> Bitmap Index Scan on gin_auth_ts (cost=0.00..45.11 rows=147 width=0) (actual time=0.032..0.032 rows=0 loops=1)
15	[...] Index Cond: (to_tsvector('english'::regconfig, (((COALESCE(username, "::character varying))::text || ' '::text) || COALESCE(description, "::text))) @@ "володимир" & "президент"::tsquery)
16	[...] -> Index Scan using author_id_conv on conversations c (cost=0.44..473.65 rows=118 width=178) (never executed)
17	[...] Index Cond: (author_id = a.id)
18	[...] -> Nested Loop (cost=58.70..10190.56 rows=809 width=277) (actual time=0.929..6.926 rows=838 loops=1)
19	[...] -> Bitmap Heap Scan on conversations c_1 (cost=58.27..3438.51 rows=809 width=178) (actual time=0.901..2.060 rows=838 loops=1)
20	[...] Recheck Cond: ("володимир" & "президент"::tsquery @@ to_tsvector('english'::regconfig, content))
21	[...] Heap Blocks: exact=823
22	[...] -> Bitmap Index Scan on gin_conv_ts (cost=0.00..58.07 rows=809 width=0) (actual time=0.756..0.756 rows=838 loops=1)
23	[...] Index Cond: (to_tsvector('english'::regconfig, content) @@ "володимир" & "президент"::tsquery)
24	[...] -> Index Scan using id_auth on authors a_1 (cost=0.43..8.35 rows=1 width=115) (actual time=0.005..0.005 rows=1 loops=838)
25	[...] Index Cond: (id = c_1.author_id)
26	Planning Time: 0.956 ms
27	Execution Time: 124.783 ms