

Pokročilé databázové technológie

Zadanie 2 – vyhľadávanie a indexovanie

Marek Adamovič

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:

1	EXPLAIN ANALYSE SELECT * FROM authors WHERE username = 'mfa_russia'
Data output Messages Notifications	
	QUERY PLAN
	text
1	Gather (cost=1000.00..147358.89 rows=1 width=125) (actual time=538.609..542.376 rows=1 loops=1)
2	Workers Planned: 2
3	Workers Launched: 2
4	-> Parallel Seq Scan on authors (cost=0.00..146358.79 rows=1 width=125) (actual time=316.689..477.608 rows=0 loops=1)
5	Filter: ((username)::text = 'mfa_russia'::text)
6	Rows Removed by Filter: 1965058
7	Planning Time: 0.067 ms
8	Execution Time: 542.399 ms

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.

1	EXPLAIN ANALYSE SELECT * FROM authors WHERE username = 'mfa_russia'
Data output Messages Notifications	
	QUERY PLAN text
1	Gather (cost=1000.00..147358.89 rows=1 width=125) (actual time=538.609..542.376 rows=1 loops=1)
2	Workers Planned: 2
3	Workers Launched: 2
4	-> Parallel Seq Scan on authors (cost=0.00..146358.79 rows=1 width=125) (actual time=316.689..477.608 rows=0 loops=1)
5	Filter: ((username)::text = 'mfa_russia'::text)
6	Rows Removed by Filter: 1965058
7	Planning Time: 0.067 ms
8	Execution Time: 542.399 ms

1	SET max_parallel_workers_per_gather = 4;
2	EXPLAIN ANALYSE SELECT * FROM authors WHERE username = 'mfa_russia'
Data output Messages Notifications	
	QUERY PLAN text
1	Gather (cost=1000.00..135074.58 rows=1 width=125) (actual time=67.252..483.569 rows=1 loops=1)
2	Workers Planned: 4
3	Workers Launched: 4
4	-> Parallel Seq Scan on authors (cost=0.00..134074.48 rows=1 width=125) (actual time=347.255..429.520 rows=0 loops=1)
5	Filter: ((username)::text = 'mfa_russia'::text)
6	Rows Removed by Filter: 1179035
7	Planning Time: 0.085 ms
8	Execution Time: 483.592 ms

```
1 SET max_parallel_workers_per_gather = 8;  
2 EXPLAIN ANALYSE SELECT * FROM authors WHERE username = 'mfa_russia'
```

Data output Messages Notifications

QUERY PLAN
text

1	Gather (cost=1000.00..131389.28 rows=1 width=125) (actual time=487.348..492.112 rows=1 loops=1)
2	Workers Planned: 5
3	Workers Launched: 5
4	-> Parallel Seq Scan on authors (cost=0.00..130389.18 rows=1 width=125) (actual time=361.486..431.271 rows=0 loops=1)
5	Filter: ((username)::text = 'mfa_russia'::text)
6	Rows Removed by Filter: 982529
7	Planning Time: 0.067 ms
8	Execution Time: 492.133 ms

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:

1	CREATE INDEX username ON authors USING btree(username);
2	EXPLAIN ANALYSE SELECT * FROM authors WHERE username = 'mfa_russia';

Data output	Messages	Notifications
-------------	----------	---------------

	QUERY PLAN
1	Index Scan using username on authors (cost=0.43..8.45 rows=1 width=125) (actual time=0.037..0.038 rows=1 loop...
2	Index Cond: ((username)::text = 'mfa_russia'::text)
3	Planning Time: 0.083 ms
4	Execution Time: 0.055 ms

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, ktorí 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:

1	EXPLAIN ANALYZE SELECT * FROM authors WHERE followers_count BETWEEN 100 AND 200
Data output Messages Notifications	
	QUERY PLAN text
1	Seq Scan on authors (cost=0.00..204075.64 rows=754975 width=125) (actual time=0.059..1136.132 rows=760088 loops=1)
2	Filter: ((followers_count >= 100) AND (followers_count <= 200))
3	Rows Removed by Filter: 5135088
4	Planning Time: 0.085 ms
5	Execution Time: 1158.326 ms

1	EXPLAIN ANALYZE SELECT * FROM authors WHERE followers_count BETWEEN 100 AND 120
Data output Messages Notifications	
	QUERY PLAN text
1	Gather (cost=1000.00..172303.05 rows=188102 width=125) (actual time=0.361..567.114 rows=199937 loops=1)
2	Workers Planned: 2
3	Workers Launched: 2
4	-> Parallel Seq Scan on authors (cost=0.00..152492.85 rows=78376 width=125) (actual time=0.251..512.758 rows=6664...)
5	Filter: ((followers_count >= 100) AND (followers_count <= 120))
6	Rows Removed by Filter: 1898413
7	Planning Time: 0.098 ms
8	Execution Time: 573.956 ms

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:

```
1 CREATE INDEX followers_count ON authors USING btree(followers_count);
```

Data output Messages Notifications

CREATE INDEX

Query returned successfully in 7 secs 443 msec.

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

```
1 EXPLAIN ANALYZE SELECT * FROM authors WHERE followers_count BETWEEN 100 AND 200;
```

Data output Messages Notifications

	QUERY PLAN	
	text	
1	Bitmap Heap Scan on authors (cost=10446.93..192676.00 rows=754975 width=125) (actual time=66.462..1169.098 rows=760088 loops...	
2	Recheck Cond: ((followers_count >= 100) AND (followers_count <= 200))	
3	Rows Removed by Index Recheck: 2922178	
4	Heap Blocks: exact=48943 lossy=66179	
5	-> Bitmap Index Scan on followers_count (cost=0.00..10258.18 rows=754975 width=0) (actual time=55.992..55.993 rows=760088 loops...	
6	Index Cond: ((followers_count >= 100) AND (followers_count <= 200))	
7	Planning Time: 0.112 ms	
8	Execution Time: 1192.292 ms	

1	EXPLAIN ANALYZE SELECT * FROM authors WHERE followers_count BETWEEN 100 AND 120;
Data output Messages Notifications	
	QUERY PLAN
	text
1	Gather (cost=3604.48..168191.01 rows=188102 width=125) (actual time=41.071..523.506 rows=199937 loops=1)
2	Workers Planned: 2
3	Workers Launched: 2
4	-> Parallel Bitmap Heap Scan on authors (cost=2604.48..148380.81 rows=78376 width=125) (actual time=13.631..431.960 rows=66646 ...
5	Recheck Cond: ((followers_count >= 100) AND (followers_count <= 120))
6	Rows Removed by Index Recheck: 544157
7	Heap Blocks: exact=24456 lossy=13401
8	-> Bitmap Index Scan on followers_count (cost=0.00..2557.45 rows=188102 width=0) (actual time=29.829..29.829 rows=199937 loops=1)
9	Index Cond: ((followers_count >= 100) AND (followers_count <= 120))
10	Planning Time: 0.110 ms
11	Execution Time: 530.027 ms

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 viac. 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:

```
1 INSERT INTO authors VALUES(-27, 'test', 'test', 'test', 27, 27, 27, 27)
```

Data output Messages Notifications

```
INSERT 0 1
```

Query returned successfully in 40 secs 906 msec.

```
1 DROP INDEX name;  
2 DROP INDEX description;  
3 DROP INDEX followers_count;
```

Data output Messages Notifications

```
DROP INDEX
```

Query returned successfully in 205 msec.

```
1 INSERT INTO authors VALUES(-28, 'test', 'test', 'test', 27, 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. Porovnajete ich dĺžku vytvárania. Prečo je tu taký rozdiel? Čím je ovplyvnená dĺžka vytvárania indexu a prečo?

Odpoveď:

```
1 CREATE INDEX IF NOT EXISTS conv_retweet_count ON conversations USING btree(retweet_count);
```

Data output Messages Notifications

CREATE INDEX

Query returned successfully in 2 min 43 secs.


```
1 CREATE INDEX IF NOT EXISTS conv_content ON conversations USING btree(content);
```

Data output Messages Notifications

CREATE INDEX

Query returned successfully in 13 min 26 secs.

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 viac miesta. Indexovanie je z veľkej časti zoradovanie a menšie dáta sa zoradujú ľahšie.

8. otázka

Otázka:

Porovnaj 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:

```
1 CREATE EXTENSION IF NOT EXISTS pgstattuple;
2
3 SELECT
4     'retweet_count' name_of_column,
5     internal_pages internal_nodes,
6     tree_level,
7     index_size / (SELECT COUNT(*) FROM conversations) avg_item_size_in_bytes
8 FROM pgstatindex('conv_retweet_count')
9 UNION
10 SELECT
11     'content' name_of_column,
12     internal_pages internal_nodes,
13     tree_level,
14     index_size / (SELECT COUNT(*) FROM conversations) avg_item_size_in_bytes
15 FROM pgstatindex('conv_content')
16 UNION
17 SELECT
18     'followers_count' name_of_column,
19     internal_pages internal_nodes,
20     tree_level,
21     index_size / (SELECT COUNT(*) FROM authors) avg_item_size_in_bytes
22 FROM pgstatindex('auth_followers_count')
23 UNION
24 SELECT
25     'name' name_of_column,
26     internal_pages internal_nodes,
27     tree_level,
28     index_size / (SELECT COUNT(*) FROM authors) avg_item_size_in_bytes
29 FROM pgstatindex('auth_name')
```

Data output Messages Notifications



	name_of_column text	internal_nodes bigint	tree_level integer	avg_item_size_in_bytes bigint
1	retweet_count	136	2	6
2	followers_count	26	2	7
3	content	12239	5	77
4	name	161	3	33

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:

1	EXPLAIN ANALYZE
2	SELECT *
3	FROM conversations
4	WHERE content LIKE '%Gates%'

Data output	Messages	Notifications
	QUERY PLAN	
	text	
1	Gather (cost=1000.00..1182831.62 rows=3056 width=220) (actual time=0.426..172362.557 rows=4199 loops=1)	
2	Workers Planned: 2	
3	Workers Launched: 2	
4	-> Parallel Seq Scan on conversations (cost=0.00..1181526.02 rows=1273 width=220) (actual time=75.530..172316.339 rows=1400 loops=3)	
5	Filter: (content ~~ '%Gates%':text)	
6	Rows Removed by Filter: 10780937	
7	Planning Time: 0.206 ms	
8	Execution Time: 172363.394 ms	

1	CREATE INDEX conv_content ON conversations USING btree(content);
---	--

Data output Messages Notifications

CREATE INDEX








Query returned successfully in 12 min 41 secs.


Slovenská Technická Univerzita v Bratislave

Fakulta Informatiky a Informačných Technológií

1	EXPLAIN ANALYZE
2	SELECT *
3	FROM conversations
4	WHERE content LIKE '%Gates%'

Data output	Messages	Notifications
-------------	----------	---------------

						
---	---	---	---	---	---	---

	QUERY PLAN	
	text	
1	Gather (cost=1000.00..1182831.62 rows=3056 width=220) (actual time=0.402..171077.088 rows=4199 loops=1)	
2	Workers Planned: 2	
3	Workers Launched: 2	
4	-> Parallel Seq Scan on conversations (cost=0.00..1181526.02 rows=1273 width=220) (actual time=56.996..171012.036 rows=1400 loops=3)	
5	Filter: (content ~~ '%Gates%':text)	
6	Rows Removed by Filter: 10780937	
7	Planning Time: 3.282 ms	
8	Execution Time: 171077.879 ms	

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:

1	EXPLAIN ANALYZE
2	SELECT *
3	FROM conversations
4	WHERE content LIKE 'There are no excuses%' AND possibly_sensitive = TRUE

Data output	Messages	Notifications
-------------	----------	---------------

+	📄	▼	📋	🗑️	🔄	⬇️	📈
---	---	---	---	----	---	----	---

	QUERY PLAN	🔒
	text	
1	Gather (cost=1000.00..1182528.92 rows=29 width=220) (actual time=104058.579..104062.408 rows=1 loops=1)	
2	Workers Planned: 2	
3	Workers Launched: 2	
4	-> Parallel Seq Scan on conversations (cost=0.00..1181526.02 rows=12 width=220) (actual time=95088.656..104028.446 rows=0 lo...	
5	Filter: (possibly_sensitive AND (content ~~ 'There are no excuses%':text))	
6	Rows Removed by Filter: 10782337	
7	Planning Time: 0.177 ms	
8	Execution Time: 104062.434 ms	

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 zefektívniť, 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.

Slovenská Technická Univerzita v Bratislave

Fakulta Informatiky a Informačných Technológií

```
1 CREATE INDEX conv_left20_content ON conversations USING btree(LEFT(content, 20));
2
3 EXPLAIN ANALYZE
4 SELECT *
5 FROM conversations
6 WHERE LEFT(content, 20) = 'There are no excuses' AND possibly_sensitive = TRUE
```

Data output Messages Notifications



	QUERY PLAN	
	text	
1	Gather (cost=3469.96..588124.51 rows=1536 width=220) (actual time=0.452..52.377 rows=1 loops=1)	
2	Workers Planned: 2	
3	Workers Launched: 2	
4	-> Parallel Bitmap Heap Scan on conversations (cost=2469.96..586970.91 rows=640 width=220) (actual time=0.060..0.061 rows=0 loops=3)	
5	Recheck Cond: ("left"(content, 20) = 'There are no excuses'::text)	
6	Filter: possibly_sensitive	
7	Rows Removed by Filter: 2	
8	Heap Blocks: exact=6	
9	-> Bitmap Index Scan on conv_left20_content (cost=0.00..2469.57 rows=161735 width=0) (actual time=0.100..0.100 rows=6 loops=1)	
10	Index Cond: ("left"(content, 20) = 'There are no excuses'::text)	
11	Planning Time: 0.109 ms	
12	Execution Time: 52.413 ms	

11. otázka

Otázka:

Vytvorte nový btree index, tak aby ste pomocou neho vedeli vyhľadať tweet, ktorý končí reťazcom „https://t.co/pkFwLXZlEm“ 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.

```
1 CREATE INDEX conv_right23_up_content ON conversations USING btree(UPPER(RIGHT(content, 23)));
2
3 EXPLAIN ANALYZE
4 SELECT *
5 FROM conversations
6 WHERE UPPER(RIGHT(content, 23)) = UPPER('https://t.co/pkFwLXZlEm');
```

Data output Messages Notifications

QUERY PLAN
text

1	Index Scan using conv_right23_up_content on conversations (cost=0.56..635386.92 rows=161735 width=516) (actual time=19.528..19.532 rows=1 loops=1)
2	Index Cond: (upper("right"(content, 23)) = 'HTTPS://T.CO/PKFWLXZLEM':text)
3	Planning Time: 5.763 ms
4	Execution Time: 19.550 ms

12. otázka









Otázka:

Nájdite conversations, ktoré majú reply_count väčší ako 150, retweet_count väčší rovný ako 5000 a výsledok zoradíte 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 EXPLAIN ANALYZE
2 SELECT *
3 FROM conversations
4 WHERE reply_count > 150 AND retweet_count >= 5000
5 ORDER BY quote_count
```

Data outputMessagesNotifications











	QUERY PLAN	
	text	
1	Gather Merge (cost=1216239.76..1216340.57 rows=864 width=220) (actual time=166787.727..166794.486 rows=8364 loops=1)	
2	Workers Planned: 2	
3	Workers Launched: 2	
4	-> Sort (cost=1215239.74..1215240.82 rows=432 width=220) (actual time=166750.990..166751.253 rows=2788 loops=3)	
5	Sort Key: quote_count	
6	Sort Method: quicksort Memory: 1199kB	
7	Worker 0: Sort Method: quicksort Memory: 1144kB	
8	Worker 1: Sort Method: quicksort Memory: 1247kB	
9	-> Parallel Seq Scan on conversations (cost=0.00..1215220.82 rows=432 width=220) (actual time=2110.585..166747.270 rows=2788 loops=3)	
10	Filter: ((reply_count > 150) AND (retweet_count >= 5000))	
11	Rows Removed by Filter: 10779549	
12	Planning Time: 4.353 ms	
13	Execution Time: 166795.008 ms	

```
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);
```

1	EXPLAIN ANALYZE
2	SELECT *
3	FROM conversations
4	WHERE reply_count > 150 AND retweet_count >= 5000
5	ORDER BY quote_count

Data output	Messages	Notifications
-------------	----------	---------------

						
---	---	---	---	---	---	---

	QUERY PLAN	
	text	
1	Sort (cost=7425.07..7427.66 rows=1037 width=220) (actual time=8916.891..8917.903 rows=8364 loops=1)	
2	Sort Key: quote_count	
3	Sort Method: quicksort Memory: 3529kB	
4	--> Index Scan using conv_reply_count on conversations (cost=0.44..7373.13 rows=1037 width=220) (actual time=86.605..8909.934 rows=8364...	
5	Index Cond: (reply_count > 150)	
6	Filter: (retweet_count >= 5000)	
7	Rows Removed by Filter: 94248	
8	Planning Time: 46.462 ms	
9	Execution Time: 8918.543 ms	

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:

```
1 CREATE INDEX conv_multi ON conversations USING btree(reply_count, retweet_count)
```

Data output Messages Notifications

CREATE INDEX

Query returned successfully in 2 min 31 secs.

```
1 EXPLAIN ANALYZE
2 SELECT *
3 FROM conversations
4 WHERE reply_count > 150 AND retweet_count >= 5000
5 ORDER BY quote_count
```

Data output Messages Notifications

QUERY PLAN

1	Sort (cost=2411.91..2414.51 rows=1037 width=220) (actual time=1868.556..1869.573 rows=8364 loops=1)
2	Sort Key: quote_count
3	Sort Method: quicksort Memory: 3529kB
4	-> Index Scan using conv_multi on conversations (cost=0.44..2359.97 rows=1037 width=220) (actual time=0.071..1863.415 rows=8364 loops=1)
5	Index Cond: ((reply_count > 150) AND (retweet_count >= 5000))
6	Planning Time: 2.417 ms
7	Execution Time: 1870.190 ms

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:

Napište 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:

```
1 ALTER TABLE conversations
2 ADD COLUMN content_tsvector tsvector;
3
4 UPDATE conversations
5 SET content_tsvector = to_tsvector(content);
6
7 CREATE INDEX conv_content_tsvector_gist ON conversations USING gist(content_tsvector);
```

Data output Messages Notifications

CREATE INDEX

Query returned successfully in 1 hr 41 min.

```
1 EXPLAIN ANALYZE SELECT * FROM conversations
2 WHERE content_tsvector @@ to_tsquery('New World Order' & Putin) AND possibly_sensitive = TRUE
```

Data output Messages Notifications

QUERY PLAN

text









1	Index Scan using conv_content_tsvector_gist on conversations (cost=0.67..8.69 rows=1 width=516) (actual time=1038.760..12301.410 rows=7 loops=1)
2	Index Cond: (content_tsvector @@ to_tsquery('New World Order' & Putin::text))
3	Rows Removed by Index Recheck: 61
4	Filter: possibly_sensitive
5	Rows Removed by Filter: 760
6	Planning Time: 8.235 ms
7	Execution Time: 12301.501 ms


Slovenská Technická Univerzita v Bratislave

Fakulta Informatiky a Informačných Technológií

1	CREATE INDEX conv_content_tsv_gin ON conversations USING gin(content_tsv);
2	
3	EXPLAIN ANALYZE SELECT * FROM conversations
4	WHERE content_tsv @@ to_tsquery('New World Order' & Putin') AND possibly_sensitive = TRUE

Data output	Messages	Notifications
-------------	----------	---------------

       	
---	--

	QUERY PLAN	
	text	
1	Bitmap Heap Scan on conversations (cost=100.26..104.52 rows=1 width=516) (actual time=46.067..49.228 rows=7 loops=1)	
2	Recheck Cond: (content_tsv @@ to_tsquery("New World Order" & Putin)::text))	
3	Rows Removed by Index Recheck: 61	
4	Filter: possibly_sensitive	
5	Rows Removed by Filter: 760	
6	Heap Blocks: exact=828	
7	-> Bitmap Index Scan on conv_content_tsv_gin (cost=0.00..100.26 rows=1 width=0) (actual time=43.763..43.764 rows=828 loops=1)	
8	Index Cond: (content_tsv @@ to_tsquery("New World Order" & Putin)::text))	
9	Planning Time: 0.183 ms	
10	Execution Time: 49.256 ms	

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 „rechecknúť“ 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:

```
1 CREATE EXTENSION pg_trgm;
2
3 CREATE INDEX links_url_trgm ON links USING gin(url gin_trgm_ops);
4
5 EXPLAIN ANALYZE SELECT * FROM links
6 WHERE url LIKE '%darujme.sk%';
```

Data output Messages Notifications

QUERY PLAN
text

1	Bitmap Heap Scan on links (cost=236.41..4490.77 rows=1085 width=459) (actual time=11.683..11.689 rows=5 loops=1)
2	Recheck Cond: ((url)::text ~~ '%darujme.sk% '::text)
3	Heap Blocks: exact=5
4	-> Bitmap Index Scan on links_url_trgm (cost=0.00..236.14 rows=1085 width=0) (actual time=11.671..11.671 rows=5 loops=1)
5	Index Cond: ((url)::text ~~ '%darujme.sk% '::text)
6	Planning Time: 0.164 ms
7	Execution Time: 11.718 ms

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 doplníme 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

```
32 -- with (russian) lemmatization--
33 ALTER TABLE conversations
34 ADD COLUMN content_tsv tsvector;
35
36 UPDATE conversations
37 SET content_tsv = to_tsvector('russian', content);
38
39 CREATE INDEX conv_content_tsv_gin ON conversations USING gin(content_tsv);
```

Slovenská Technická Univerzita v Bratislave

Fakulta Informatiky a Informačných Technológií

51	--select query (with lem)--
52	EXPLAIN ANALYZE
53	SELECT *
54	FROM conversations con
55	JOIN authors aut ON con.author_id = aut.id AND con.content_tsv @@ to_tsquery('english', 'Володимир & Президент')
56	UNION
57	SELECT NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, *
58	FROM authors
59	WHERE multi_tsv @@ to_tsquery('english', 'Володимир & Президент')
60	ORDER BY reply_count DESC
Data output Messages Notifications	
QUERY PLAN	
text	
1	Sort (cost=10262.94..10264.96 rows=810 width=1289) (actual time=8.210..8.259 rows=1060 loops=1)
2	Sort Key: con.reply_count DESC
3	Sort Method: quicksort Memory: 1493kB
4	--> Unique (cost=10179.26..10223.81 rows=810 width=1289) (actual time=6.940..7.657 rows=1060 loops=1)
5	--> Sort (cost=10179.26..10181.28 rows=810 width=1289) (actual time=6.939..7.012 rows=1060 loops=1)
6	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...
7	Sort Method: quicksort Memory: 1493kB
8	--> Append (cost=58.70..10140.13 rows=810 width=1289) (actual time=0.623..5.735 rows=1060 loops=1)
9	--> Nested Loop (cost=58.70..10079.96 rows=809 width=514) (actual time=0.622..5.636 rows=1058 loops=1)
10	--> 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)
11	Recheck Cond: (content_tsv @@ "Володимир" & "президент":tsquery)
12	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)
14	Index Cond: (content_tsv @@ "Володимир" & "президент":tsquery)
15	--> Index Scan using authors_pkey on authors aut (cost=0.43..8.41 rows=1 width=262) (actual time=0.003..0.003 rows=1 loops=1058)
16	Index Cond: (id = con.author_id)
17	--> Bitmap Heap Scan on authors (cost=44.01..48.02 rows=1 width=431) (actual time=0.031..0.033 rows=2 loops=1)
18	Recheck Cond: (multi_tsv @@ "Володимир" & "президент":tsquery)
19	Heap Blocks: exact=2
20	--> 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)
21	Index Cond: (multi_tsv @@ "Володимир" & "президент":tsquery)
22	Planning Time: 0.437 ms
23	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--
2  ALTER TABLE conversations
3  ADD COLUMN content_tsv_en tsvector;
4
5  UPDATE conversations
6  SET content_tsv_en = to_tsvector('english', content);
7
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
15 SET multi_tsv_en = to_tsvector('english', description || ' ' || username);
16
17 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 skľuzu 🐼)