

**Slovenská Technická Univerzita**  
**Fakulta Informatiky a Informačných technológií**

**Pokročilé Databázové Technológie**  
**Zadanie 2**

## Contents

Zadanie .....	3
Úloha č.1 .....	5
Úloha č.2 .....	6
Úloha č.3 .....	8
Úloha č.4 .....	9
Úloha č.5 .....	10
Úloha č.6 .....	11
Úloha č.7 .....	12
Úloha č.8 .....	14
Úloha č.9 .....	15
Úloha č.10 .....	17
Úloha č.11 .....	18
Úloha č.12 .....	19
Úloha č.13 .....	20
Úloha č.14 .....	23
Úloha č.15 .....	25
Úloha č.16 .....	27
Úloha č.17 .....	28
Úloha č.18 .....	30

## Zadanie

### Zadanie 2 – vyhľadávanie a indexovanie

Odovzdanie do 24.10.2021 23:59 – máte na to 2 týždne – dostanete za to 7,5 boda.

Otázky 2-17 sú dokopy za 5,5 boda (každá rovnako). Zadanie 18 je za 2 body. Teda good luck and have fun.

Zadania prosím neopisujte jednoslovné ale zmysluplnou vetou (nie slohová práca - teda vecne). Vždy priložte screenshot z explain analyse.

1. Vyhľadajte v accounts screen\_name s presnou hodnotou 'realDonaldTrump' a analyzujte daný select. Akú metódu vám vybral plánovač a prečo - odôvodnite prečo sa rozhodol tak ako sa rozhodol?

2. 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?

3. Vytvorte btree index nad screen\_name 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?

4. Vyberte používateľov, ktorí majú followers\_count väčší, rovný ako 100 a zároveň menší, rovný 200. Je správanie rovnaké v prvej úlohe? Je správanie rovnaké ako v tretej úlohe? Prečo?

5. Vytvorte index nad 4 úlohou a popíšte prácu s indexom. Čo je to Bitmap Index Scan a prečo je tam Bitmap Heap Scan? Prečo je tam recheck condition?

6. Vyberte používateľov, ktorí majú followers\_count väčší, rovný ako 100 a zároveň menší, rovný 1000? V čom je rozdiel, prečo?

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

8. Vytvorte btree index nad tweetami pre retweet\_count a pre content. Porovnajte ich dĺžku vytvárania. Prečo je tu taký rozdiel? Čím je ovplyvnená dĺžka vytvárania indexu a prečo?

9. Porovnajte indexy pre retweet\_count, content, followers\_count, screen\_name,... v čom sa líšia a prečo (opíšte výstupné hodnoty pre všetky indexy)?

a. create extension pageinspect;

b. select \* from bt\_metap('idx\_content');

c. select type, live\_items, dead\_items, avg\_item\_size, page\_size, free\_size from

bt\_page\_stats('idx\_content',1000);

d. select \* from bt\_page\_items('idx\_content',1) limit 1000;

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

11. Vyhľadajte tweet, ktorý začína “The Cabel and Deep State”. Použil sa index?

12. Teraz naindexujte content tak, aby sa použil btree index a zhodnoťte prečo sa pred tým nad “The Cabel and Deep State” nepoužil. Použije sa teraz na „Gates“ na ľubovoľnom mieste? Zdôvodnite použitie alebo nepoužitie indexu?

13. Vytvorte nový btree index, tak aby ste pomocou neho vedeli vyhľadať tweet, ktorý končí reťazcom „idiot #QAnon“ kde nezáleží na tom ako to napíšete. Popíšte čo jednotlivé funkcie robia.

14. Nájdite účty, ktoré majú follower\_count menší ako 10 a friends\_count väčší ako 1000 a výsledok zoradte podľa statuses\_count. Následne spravte jednoduché indexy a popíšte ktoré má a ktoré nemá zmysel robiť a prečo.

15. Na predošlú query spravte zložený index a porovnajte výsledok s tým, keď je sú indexy separátne. Výsledok zdôvodnite.

16. Upravte query tak, aby bol follower\_count menší ako 1000 a friends\_count väčší ako 1000. V čom je rozdiel a prečo?

17. Vytvorte vhodný index pre vyhľadávanie písmen bez kontextu nad screen\_name v accounts. Porovnajte výsledok pre vyhľadanie presne ‘realDonaldTrump’ voči btree indexu? Ktorý index sa vybral a prečo? Následne vyhľadajte v texte screen\_name ‘ldonaldt’ a porovnajte výsledky. Aký index sa vybral a prečo?

18. Vytvorte query pre slová "John" a "Oliver" pomocou FTS (tsvector a tsquery) v angličtine v stĺpcoch tweets.content, accounts.decription a accounts.name, kde slová sa môžu nachádzať v prvom, druhom ALEBO treťom stĺpci. Teda vyhovujúci záznam je ak aspoň jeden stĺpec má „match“. Výsledky zoradte 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.

## Úloha č.1

Selectom:

```
EXPLAIN ANALYZE SELECT * FROM accounts WHERE screen_name = 'realDonaldTrump'
```

	QUERY PLAN	
	text	
1	Gather (cost=1000.00..223590.46 rows=1 width=119) (actual time=3.484..510.712 rows=1 loops=1)	
2	[...] Workers Planned: 2	
3	[...] Workers Launched: 2	
4	[...] -> Parallel Seq Scan on accounts (cost=0.00..222590.36 rows=1 width=119) (actual time=307.637..475.583 rows=0 loops=3)	
5	[...] Filter: ((screen_name)::text = 'realDonaldTrump'::text)	
6	[...] Rows Removed by Filter: 3229079	
7	Planning Time: 0.094 ms	
8	JIT:	
9	[...] Functions: 6	
10	[...] Options: Inlining false, Optimization false, Expressions true, Deforming true	
11	[...] Timing: Generation 1.523 ms, Inlining 0.000 ms, Optimization 0.732 ms, Emission 8.066 ms, Total 10.320 ms	
12	Execution Time: 511.427 ms	

Z tohto môžeme vidieť, že plánovač si vybral paralelný sekvenčný sken. Paralelný je preto, pretože je výhodnejšie si počet záznamov rozdeliť medzi všetkých workerov a tým sa vlastne skrátí čas potrebný na prejdienie všetkých záznamov. Sekvenčný je zasa preto, pretože sa nemá planner čoho chytiť. Nie je tam index, podľa ktorého by dokázal množinu záznamov vylúčiť, teda musí prejsť všetky

## Úloha č.2

Na tomto selecte pracovali 2 workeri. Workeri slúžia na rozdelenie si práce na selecte. Podobne ako thready pri multithreadovom vykonávaní. Zvýšením počtu workerov sa zníži čas vykonávania, pretože viacero vláken procesora môže vykonávať paralelnú prácu. Je tam ale strop, zrejme počet vláken procesora na ktorom to beží. V tom pridanie ďalších workerov nezníži dobu vykonávania ale zvýši réžiu spojenú s rozdelením a spojením práce.

Počet workerov som zdvihol pomocou SET max\_parallel\_workers\_per\_gather = 4;

1	Gather (cost=1000.00..223590.46 rows=1 width=119) (actual time=7.346..746.057 rows=1 loops=1)
2	[...] Workers Planned: 2
3	[...] Workers Launched: 2
4	[...] -> Parallel Seq Scan on accounts (cost=0.00..222590.36 rows=1 width=119) (actual time=439.997..684.667 rows=0 loops=3)
5	[...] Filter: ((screen_name)::text = 'realDonaldTrump'::text)
6	[...] Rows Removed by Filter: 3229079
7	Planning Time: 0.118 ms
8	JIT:
9	[...] Functions: 6
10	[...] Options: Inlining false, Optimization false, Expressions true, Deforming true
11	[...] Timing: Generation 2.794 ms, Inlining 0.000 ms, Optimization 2.121 ms, Emission 17.855 ms, Total 22.769 ms
12	Execution Time: 746.989 ms

	QUERY PLAN	
	text	
1	Gather (cost=1000.00..203408.72 rows=1 width=119) (actual time=10.045..634.994 rows=1 loops=1)	
2	[...] Workers Planned: 4	
3	[...] Workers Launched: 4	
4	[...] -> Parallel Seq Scan on accounts (cost=0.00..202408.62 rows=1 width=119) (actual time=401.272..525.190 rows=0 loops=5)	
5	[...] Filter: ((screen_name)::text = 'realDonaldTrump'::text)	
6	[...] Rows Removed by Filter: 1937447	
7	Planning Time: 0.135 ms	
8	JIT:	
9	[...] Functions: 10	
10	[...] Options: Inlining false, Optimization false, Expressions true, Deforming true	
11	[...] Timing: Generation 11.119 ms, Inlining 0.000 ms, Optimization 3.231 ms, Emission 47.670 ms, Total 62.021 ms	
12	Execution Time: 635.692 ms	

	<div> <div>QUERY PLAN</div> <div>text</div> </div> <div></div>
1	Gather (cost=1000.00..197354.20 rows=1 width=119) (actual time=6.676..658.945 rows=1 loops=1)
2	[...] Workers Planned: 5
3	[...] Workers Launched: 5
4	[...] -> Parallel Seq Scan on accounts (cost=0.00..196354.10 rows=1 width=119) (actual time=450.610..558.265 rows=0 loops=6)
5	[...] Filter: ((screen_name)::text = 'realDonaldTrump'::text)
6	[...] Rows Removed by Filter: 1614540
7	Planning Time: 0.131 ms
8	JIT:
9	[...] Functions: 12
10	[...] Options: Inlining false, Optimization false, Expressions true, Deforming true
11	[...] Timing: Generation 6.858 ms, Inlining 0.000 ms, Optimization 5.367 ms, Emission 82.140 ms, Total 94.365 ms
12	Execution Time: 659.558 ms

	<div> <div>QUERY PLAN</div> <div>text</div> </div> <div></div>
1	Gather (cost=1000.00..197354.20 rows=1 width=119) (actual time=8.971..598.382 rows=1 loops=1)
2	[...] Workers Planned: 5
3	[...] Workers Launched: 5
4	[...] -> Parallel Seq Scan on accounts (cost=0.00..196354.10 rows=1 width=119) (actual time=392.358..489.403 rows=0 loops=6)
5	[...] Filter: ((screen_name)::text = 'realDonaldTrump'::text)
6	[...] Rows Removed by Filter: 1614540
7	Planning Time: 0.154 ms
8	JIT:
9	[...] Functions: 12
10	[...] Options: Inlining false, Optimization false, Expressions true, Deforming true
11	[...] Timing: Generation 5.696 ms, Inlining 0.000 ms, Optimization 3.906 ms, Emission 49.136 ms, Total 58.737 ms
12	Execution Time: 599.237 ms

## Úloha č.3

CREATE INDEX idx\_screen\_name ON accounts USING btree (screen\_name);

CREATE INDEX

Query returned successfully in 51 secs 842 msec.

EXPLAIN ANALYZE SELECT \* FROM accounts WHERE screen\_name = 'realDonaldTrump'


QUERY PLAN		
text		
1	Index Scan using idx_screen_name on accounts (cost=0.43..8.45 rows=1 width=119) (actual time=0.081..0.083 rows=1 loops=1)	
2	[...] Index Cond: ((screen_name)::text = 'realDonaldTrump'::text)	
3	Planning Time: 0.247 ms	
4	Execution Time: 0.099 ms	

Čas sa výrazným spôsobom zmenšil, pretože query nepoužíva sekvenčné prechádzanie záznamov ale index kedy nemusí prejsť všetky záznamy. V takomto prípade query nepoužíva workerov ale iba jeden proces prechádza indexom.



## Úloha č.4

EXPLAIN ANALYZE SELECT \* FROM accounts WHERE followers\_count >= 100 AND followers\_count <= 200

	QUERY PLAN	
	text	
1	Seq Scan on accounts (cost=0.00..317444.57 rows=1232038 width=119) (actual time=4.337..1250.423 rows=1269496 loops=1)	
2	[...] Filter: ((followers_count >= 100) AND (followers_count <= 200))	
3	[...] Rows Removed by Filter: 8417742	
4	Planning Time: 0.106 ms	
5	JIT:	
6	[...] Functions: 2	
7	[...] Options: Inlining false, Optimization false, Expressions true, Deforming true	
8	[...] Timing: Generation 0.855 ms, Inlining 0.000 ms, Optimization 0.331 ms, Emission 3.804 ms, Total 4.991 ms	
9	Execution Time: 1291.919 ms	

Správanie nie je rovnaké ako v prvej úlohe, pretože tam použil plánovač paralelný sekvenčný sken a v úlohe 3 tam bol zasa sekvenčný sken. Nepoužil paralelný kvôli tomu, že vybraných záznamov je výrazne viac ako v prvej úlohe a indexy sú pomalé pre výber viacerých záznamov, pretože indexom vytiahneme jeden vyhovujúci a potom musíme hľadať ďalší a ísť indexom znova.

## Úloha č.5

CREATE INDEX idx\_followers\_count ON accounts (followers\_count);

```
CREATE INDEX
```

```
Query returned successfully in 6 secs 229 msec.
```


EXPLAIN ANALYZE SELECT \* FROM accounts WHERE followers\_count >= 100 AND followers\_count <= 200

	QUERY PLAN text	
1	Bitmap Heap Scan on accounts (cost=16944.82..310246.28 rows=1232038 width=119) (actual time=112.052..1657.642 rows=126...	
2	[...] Recheck Cond: ((followers_count >= 100) AND (followers_count <= 200))	
3	[...] Rows Removed by Index Recheck: 6449000	
4	[...] Heap Blocks: exact=39770 lossy=132186	
5	[...] -> Bitmap Index Scan on idx_followers_count (cost=0.00..16636.82 rows=1232038 width=0) (actual time=100.869..100.869 ro...	
6	[...] Index Cond: ((followers_count >= 100) AND (followers_count <= 200))	
7	Planning Time: 0.315 ms	
8	JIT:	
9	[...] Functions: 2	
10	[...] Options: Inlining false, Optimization false, Expressions true, Deforming true	
11	[...] Timing: Generation 0.644 ms, Inlining 0.000 ms, Optimization 0.255 ms, Emission 2.283 ms, Total 3.182 ms	
12	Execution Time: 1697.500 ms	

Bitmap Heap scan - Planer najprv prejde indexom a spojí všetky riadky a bloky, v ktorých sa nachádza splnená podmienka WHERE. Pomocou tohto listu potom hľadá všetky tieto záznamy. Recheck je tam na to keď mu dôjde pracovná pamäť, tak neuchováva jeden bit pre riadok ale pre celú stránku. Vtedy je hu potrebné znovu overiť.

## Úloha č.6

EXPLAIN ANALYZE SELECT \* FROM accounts WHERE followers\_count >= 100 AND followers\_count <= 1000

QUERY PLAN		
	text	
1	Seq Scan on accounts (cost=0.00..317444.57 rows=4395042 width=119) (actual time=3.547..1386.163 rows=4382646 loops=1)	
2	[...] Filter: ((followers_count >= 100) AND (followers_count <= 1000))	
3	[...] Rows Removed by Filter: 5304592	
4	Planning Time: 0.093 ms	
5	JIT:	
6	[...] Functions: 2	
7	[...] Options: Inlining false, Optimization false, Expressions true, Deforming true	
8	[...] Timing: Generation 0.530 ms, Inlining 0.000 ms, Optimization 0.340 ms, Emission 3.014 ms, Total 3.885 ms	
9	Execution Time: 1527.117 ms	

Rozdiel je v tom, že v tomto prípade planer nepoužil index, ktorý som vytvoril nad týmto stĺpcom, pretože planer asi vypočítal, že réžia by bola ešte vyššia ako predtým a keďže vytiahol ešte viac záznamov, tak by potreboval ešte viac pamäte, recheckov a času na vykonanie tohto selectu. Preto použil sekvenčný sken. Už ani v úlohe č.5 nám index nepomohol, pretože to trvalo dlhšie ako bez indexu.

## Úloha č.7

CREATE INDEX idx\_name ON accounts USING btree (name);

```
CREATE INDEX
```

Query returned successfully in 2 min 7 secs.

CREATE INDEX idx\_friends\_count ON accounts USING btree (friends\_count);

```
CREATE INDEX
```

Query returned successfully in 9 secs 403 msec.


CREATE INDEX idx\_description ON accounts USING btree (description);

```
CREATE INDEX
```

Query returned successfully in 3 min 6 secs.

INSERT INTO accounts(screen\_name, name, description, followers\_count, friends\_count, statuses\_count)

VALUES ('Test123', 'Test123', 'Moj popis', 123, 321, 132)

QUERY PLAN		
	text	
1	Insert on accounts (cost=0.00..0.01 rows=1 width=888) (actual time=0.268..0.269 rows=0 loops=1)	
2	[...] -> Result (cost=0.00..0.01 rows=1 width=888) (actual time=0.033..0.034 rows=1 loops=1)	
3	Planning Time: 0.064 ms	
4	Execution Time: 0.324 ms	

DROP INDEX idx\_name

```
DROP INDEX
```

Query returned successfully in 210 msec.

DROP INDEX idx\_friends\_count

---

DROP INDEX

Query returned successfully in 159 msec.

DROP INDEX idx\_description

---

DROP INDEX

Query returned successfully in 4 secs 191 msec.

DELETE FROM accounts WHERE id = 1


---

DELETE 0

Query returned successfully in 3 secs 25 msec.

EXPLAIN ANALYZE INSERT INTO accounts(screen\_name, name, description, followers\_count, friends\_count, statuses\_count)

VALUES ('Test123', 'Test123', 'Moj popis', 123, 321, 132)

QUERY PLAN		
	text	
1	Insert on accounts (cost=0.00..0.01 rows=1 width=888) (actual time=0.387..0.389 rows=0 loops=1)	
2	[...] -> Result (cost=0.00..0.01 rows=1 width=888) (actual time=0.039..0.040 rows=1 loops=1)	
3	Planning Time: 0.072 ms	
4	Execution Time: 0.441 ms	

Z výsledkov vyššie môžeme vidieť, že vloženie záznamu išlo o štvrtinu rýchlejšie s indexami, čo ale nie je pravidlo. Zrejme to bolo tým, že vkladané hodnoty sa rýchlo dostali na svoje miesta v btree indexoch. Pri vkladaní väčšieho množstva záznamov je vhodné dropnúť indexy, vložiť záznamy a potom ich znova vytvoriť.

## Úloha č.8

```
CREATE INDEX idx_retweet_count ON tweets USING btree (retweet_count);
```

```
CREATE INDEX
```

```
Query returned successfully in 47 secs 227 msec.
```

```
CREATE INDEX idx_content ON tweets USING btree (content);
```

```
CREATE INDEX
```

```
Query returned successfully in 6 min 24 secs.
```

Délka je ovplyvnená tým, že sa vytvára nad stringom a ten prvý nad číslom.

## Úloha č.9

create extension pageinspect;

```
CREATE EXTENSION
```

Query returned successfully in 274 msec.

Štruktúra bt\_metap

```
typedef struct BTMetaPageData
{
    uint32      btm_magic;      /* should contain BTREE_MAGIC */
    uint32      btm_version;    /* should contain BTREE_VERSION */
    BlockNumber btm_root;      /* current root location */
    uint32      btm_level;      /* tree level of the root page */
    BlockNumber btm_fastroot;   /* current "fast" root location */
    uint32      btm_fastlevel; /* tree level of the "fast" root page */
} BTMetaPageData;
```

select \* from bt\_metap('idx\_content');

	magic integer	version integer	root bigint	level bigint	fastroot bigint	fastlevel bigint	oldest_xact xid	last_cleanup_num_tuples double precision	allequalimage boolean
1	340322	4	287222	5	287222	5	0	-1	true

select \* from bt\_metap('idx\_retweet\_count');

	magic integer	version integer	root bigint	level bigint	fastroot bigint	fastlevel bigint	oldest_xact xid	last_cleanup_num_tuples double precision	allequalimage boolean
1	340322	4	209	2	209	2	0	-1	true

select \* from bt\_metap('idx\_followers\_count');

	magic integer	version integer	root bigint	level bigint	fastroot bigint	fastlevel bigint	oldest_xact xid	last_cleanup_num_tuples double precision	allequalimage boolean
1	340322	4	209	2	209	2	0	-1	true

select \* from bt\_metap('idx\_screen\_name');

	magic integer	version integer	root bigint	level bigint	fastroot bigint	fastlevel bigint	oldest_xact xid	last_cleanup_num_tuples double precision	allequalimage boolean
1	340322	4	222	2	222	2	0	-1	true

Root je pri kazdom cislo stránky, kde sa nachádza root. Level je hĺbka btree, v ktorej je root a fast level a root su tie isté veci len na uzle v strome, z ktorého sa vetví. Rozdiel medzi root a fastroot je len zopar node, ktoré sa nevetvia len ukazujú na seba.

```
select type, live_items, dead_items, avg_item_size, page_size, free_size from
```

```
bt_page_stats('idx_content',1000);
```

	type "char" (1)	live_items integer	dead_items integer	avg_item_size integer	page_size integer	free_size integer
1	I	35	0	205	8192	816

Type je typ stránky, počet itemo, ktoré sa nachádzajú na tejto stránke pre tento index. Priemerná veľkosť a nastavená veľkosť stránky a voľné miesto, ktoré sa na nej ešte nachádza.

```
select * from bt_page_items('idx_content',1) limit 1000;
```

	itemoffset smallint	ctid tid	itemlen smallint	nulls boolean	vars boolean	data text
1	1	(609623,1)	24	false	true	13 0d 0a 0d 0a 23 0d 0a 23 00 00 00 00 00 00
2	2	(16,8324)	808	false	true	03 00 00 00 00 00 00 00
3	3	(16,8265)	456	false	true	03 00 00 00 00 00 00 00
4	4	(922877,26)	16	false	true	0b 0a 0a 0a 0a 00 00 00
5	5	(788187,25)	24	false	true	19 0a 0a 0a 0a 23 20 23 20 23 20 00 00 00 00
6	6	(1039133,23)	16	false	true	09 0a 20 20 00 00 00 00

Offset stránok, ich Idčka a dĺžka stránky a data.



## Úloha č.10

### Bez indexu

EXPLAIN ANALYZE SELECT \* FROM tweets WHERE content ILIKE ('%Gates%')

	QUERY PLAN	
	text	
1	Gather (cost=1000.00..1247736.13 rows=11562 width=312) (actual time=57.113..35393.771 rows=108211 loops=1)	
2	[...] Workers Planned: 2	
3	[...] Workers Launched: 2	
4	[...] -> Parallel Seq Scan on tweets (cost=0.00..1245579.93 rows=4818 width=312) (actual time=483.835..34880.590 rows=36070 loops=3)	
5	[...] Filter: (content ~~* '%Gates%':text)	
6	[...] Rows Removed by Filter: 10622613	
7	Planning Time: 1.008 ms	
8	JIT:	
9	[...] Functions: 6	
10	[...] Options: Inlining true, Optimization true, Expressions true, Deforming true	
11	[...] Timing: Generation 1.993 ms, Inlining 1326.258 ms, Optimization 85.700 ms, Emission 34.674 ms, Total 1448.626 ms	
12	Execution Time: 35406.210 ms	

CREATE INDEX idx\_content ON tweets USING btree (content);

### S vytvoreným indexom

EXPLAIN ANALYZE SELECT \* FROM tweets WHERE content ILIKE ('%Gates%')

	QUERY PLAN	
	text	
1	Gather (cost=1000.00..1247736.13 rows=11562 width=312) (actual time=49.019..44711.695 rows=108211 loops=1)	
2	[...] Workers Planned: 2	
3	[...] Workers Launched: 2	
4	[...] -> Parallel Seq Scan on tweets (cost=0.00..1245579.93 rows=4818 width=312) (actual time=81.342..44628.977 rows=36070 loops=3)	
5	[...] Filter: (content ~~* '%Gates%':text)	
6	[...] Rows Removed by Filter: 10622613	
7	Planning Time: 1.010 ms	
8	JIT:	
9	[...] Functions: 6	
10	[...] Options: Inlining true, Optimization true, Expressions true, Deforming true	
11	[...] Timing: Generation 1.908 ms, Inlining 131.247 ms, Optimization 67.791 ms, Emission 37.082 ms, Total 238.028 ms	
12	Execution Time: 44724.627 ms	

Z výsledkov si môžeme všimnúť, že trvanie selectu s indexom bolo dlhšie. Toto je zapríčinené zrejme tým, že sa všetky data ako aj index nezmestili do pamäte a bolo ich treba čítať z disku, aj keď sa index nepoužil.

## Úloha č.11

Prvýkrát som spustil select pomocou ILIKE ale až neskôr som sa dočítal, že Btree funguje pre takýto typ select len s LIKE. Čo som si ale všimol, že to bolo oveľa rýchlejšie iba s LIKE ale index aj tak nepoužilo. To zrejme preto, že som tento index nevytvoril s locale C. podľa dokumentácie postgres 9.2 index types.

EXPLAIN ANALYZE SELECT \* FROM tweets WHERE content ILIKE ('The Cabel and Deep State%')

	QUERY PLAN text	
1	Gather (cost=1000.00..1246883.43 rows=3035 width=312) (actual time=33671.820..33676.853 rows=1 loops=1)	
2	[...] Workers Planned: 2	
3	[...] Workers Launched: 2	
4	[...] -> Parallel Seq Scan on tweets (cost=0.00..1245579.93 rows=1265 width=312) (actual time=30974.430..33627.641 rows=0 loops=3)	
5	[...] Filter: (content ~~* 'The Cabel and Deep State%':text)	
6	[...] Rows Removed by Filter: 10658683	
7	Planning Time: 0.710 ms	
8	JIT:	
9	[...] Functions: 6	
10	[...] Options: Inlining true, Optimization true, Expressions true, Deforming true	
11	[...] Timing: Generation 1.598 ms, Inlining 134.062 ms, Optimization 66.540 ms, Emission 30.158 ms, Total 232.358 ms	
12	Execution Time: 33677.415 ms	

EXPLAIN ANALYZE SELECT \* FROM tweets WHERE content LIKE ('The Cabel and Deep State%')

	QUERY PLAN text	
1	Gather (cost=1000.00..1246883.43 rows=3035 width=312) (actual time=2121.147..2125.759 rows=1 loops=1)	
2	[...] Workers Planned: 2	
3	[...] Workers Launched: 2	
4	[...] -> Parallel Seq Scan on tweets (cost=0.00..1245579.93 rows=1265 width=312) (actual time=1916.718..2078.018 rows=0 loops=3)	
5	[...] Filter: (content ~~ 'The Cabel and Deep State%':text)	
6	[...] Rows Removed by Filter: 10658683	
7	Planning Time: 1.285 ms	
8	JIT:	
9	[...] Functions: 6	
10	[...] Options: Inlining true, Optimization true, Expressions true, Deforming true	
11	[...] Timing: Generation 1.963 ms, Inlining 174.787 ms, Optimization 84.205 ms, Emission 38.026 ms, Total 298.980 ms	
12	Execution Time: 2126.359 ms	

## Úloha č.12

DROP INDEX idx\_content

```
DROP INDEX
```

```
Query returned successfully in 299 msec.
```

CREATE INDEX idx\_content ON tweets (content text\_pattern\_ops);

```
CREATE INDEX
```

```
Query returned successfully in 2 min 2 secs.
```

Popis prečo sa asi nepoužil index je v [úlohe č.11](#)

EXPLAIN ANALYZE SELECT \* FROM tweets WHERE content ILIKE ('%Gates%')

	QUERY PLAN	
	text	
1	Gather (cost=1000.00..1247736.13 rows=11562 width=312) (actual time=274.138..81677.544 rows=108211 loops=1)	
2	[...] Workers Planned: 2	
3	[...] Workers Launched: 2	
4	[...] -> Parallel Seq Scan on tweets (cost=0.00..1245579.93 rows=4818 width=312) (actual time=196.437..81526.100 rows=36070 loops=3)	
5	[...] Filter: (content ~~~ '%Gates%':text)	
6	[...] Rows Removed by Filter: 10622613	
7	Planning Time: 18.642 ms	
8	JIT:	
9	[...] Functions: 6	
10	[...] Options: Inlining true, Optimization true, Expressions true, Deforming true	
11	[...] Timing: Generation 44.581 ms, Inlining 262.167 ms, Optimization 139.016 ms, Emission 129.212 ms, Total 574.976 ms	
12	Execution Time: 81734.730 ms	

EXPLAIN ANALYZE SELECT \* FROM tweets WHERE content LIKE ('The Cabel and Deep State%')

	QUERY PLAN	
	text	
1	Index Scan using idx_content on tweets (cost=0.81..8.83 rows=3035 width=312) (actual time=0.479..0.481 rows=1 loops=1)	
2	[...] Index Cond: ((content ~>=~ 'The Cabel and Deep State':text) AND (content ~<~ 'The Cabel and Deep Statf':text))	
3	[...] Filter: (content ~~ 'The Cabel and Deep State%':text)	
4	Planning Time: 7.372 ms	
5	Execution Time: 0.511 ms	

Pri Gatesovi sa index nepoužil, pretože ak je hodnota hocikde v texte, tak sa index nedá použiť. Pri druhom príklade sa ale použil, pretože string začínal vždy rovnako, a výrazne sa zrýchlilo vykonávanie.

## Úloha č.13

DROP INDEX idx\_content

```
DROP INDEX
```


```
Query returned successfully in 297 msec.
```

CREATE INDEX idx\_content ON tweets(lower('idiot #QAnon') in lower(content))

```
CREATE INDEX
```

```
Query returned successfully in 50 secs 111 msec.
```

EXPLAIN ANALYZE SELECT \* FROM tweets WHERE content LIKE '%idiot #QAnon'

QUERY PLAN		
	text	
1	Seq Scan on tweets (cost=0.00..1638618.88 rows=10658683 width=312) (actual time=62942.643..62942.644 rows=0 loops=1)	
2	[...] Filter: ("position"(lower(content), 'idiot #qanon':text) > 0)	
3	[...] Rows Removed by Filter: 31976050	
4	Planning Time: 2.662 ms	
5	JIT:	
6	[...] Functions: 2	
7	[...] Options: Inlining true, Optimization true, Expressions true, Deforming true	
8	[...] Timing: Generation 0.735 ms, Inlining 31.532 ms, Optimization 28.850 ms, Emission 15.702 ms, Total 76.819 ms	
9	Execution Time: 62943.454 ms	

CREATE INDEX idx\_content ON tweets USING btree (lower(content));

```
DROP INDEX
```

```
Query returned successfully in 144 msec.
```

EXPLAIN ANALYZE SELECT \* FROM tweets WHERE lower(content) LIKE lower('%idiot #QAnon')

	QUERY PLAN text	
1	Gather (cost=1000.00..1280208.11 rows=3198 width=312) (actual time=28587.127..28591.955 rows=0 loops=1)	
2	[...] Workers Planned: 2	
3	[...] Workers Launched: 2	
4	[...] -> Parallel Seq Scan on tweets (cost=0.00..1278888.31 rows=1332 width=312) (actual time=28545.606..28545.607 rows=0 loops=3)	
5	[...] Filter: (lower(content) ~~ '%idiot #qanon':text)	
6	[...] Rows Removed by Filter: 10658683	
7	Planning Time: 0.280 ms	
8	JIT:	
9	[...] Functions: 6	
10	[...] Options: Inlining true, Optimization true, Expressions true, Deforming true	
11	[...] Timing: Generation 1.565 ms, Inlining 121.279 ms, Optimization 83.883 ms, Emission 38.468 ms, Total 245.195 ms	
12	Execution Time: 28592.773 ms	

CREATE INDEX idx\_content ON tweets(lower(content) varchar\_pattern\_ops)

CREATE INDEX

Query returned successfully in 4 min 34 secs.

EXPLAIN ANALYZE SELECT \* FROM tweets WHERE lower(content) LIKE lower('%idiot #QAnon')

	QUERY PLAN text	
1	Gather (cost=1000.00..1280208.11 rows=3198 width=312) (actual time=66343.237..66365.425 rows=0 loops=1)	
2	[...] Workers Planned: 2	
3	[...] Workers Launched: 2	
4	[...] -> Parallel Seq Scan on tweets (cost=0.00..1278888.31 rows=1332 width=312) (actual time=66243.603..66243.604 rows=0 loops=3)	
5	[...] Filter: (lower(content) ~~ '%idiot #qanon':text)	
6	[...] Rows Removed by Filter: 10658683	
7	Planning Time: 9.246 ms	
8	JIT:	
9	[...] Functions: 6	
10	[...] Options: Inlining true, Optimization true, Expressions true, Deforming true	
11	[...] Timing: Generation 35.347 ms, Inlining 252.365 ms, Optimization 110.195 ms, Emission 75.146 ms, Total 473.052 ms	
12	Execution Time: 66399.679 ms	

Vidno, že ani jeden z týchto nápadov nefunguje. Rozmýšlal som ako to opraviť a spomenul som si z dokumentácie a z ostatných úloh, že LIKE a btree keď percento je na konci funguju celkom pekne. Preto som tak rozmýšlal ako ten problem otočiť a našiel som, že je možné v postgrese otočiť string a indexovať ho tak, tým sa percento dostane na koniec. Spraviť tým v princípe vyhľadávanie od konca.

```
CREATE INDEX idx_content_btree ON tweets USING btree (reverse(lower(content)));
```

```
CREATE INDEX
```

```
Query returned successfully in 7 min 4 secs.
```


- Tak ani tento sa ešte nepoužil, tak skúsím na matchovanie patternu v texte ako v inej úlohe, kde to pomohlo

```
CREATE INDEX idx_content_btree ON tweets USING btree (reverse(lower(content))  
text_pattern_ops);
```

```
CREATE INDEX
```

```
Query returned successfully in 3 min 55 secs.
```

```
EXPLAIN SELECT * FROM tweets WHERE reverse(lower(content)) LIKE reverse(lower('%idiot  
#QAnon'))
```

QUERY PLAN		
	text	
1	Index Scan using idx_content_btree on tweets (cost=0.81..607757.51 rows=159880 width=312) (actual time=5.386..5.387 rows...	
2	[...] Index Cond: ((reverse(lower(content)) ~>= ~ 'nonaq# toidi':text) AND (reverse(lower(content)) ~<= ~ 'nonaq# toidj':text))	
3	[...] Filter: (reverse(lower(content)) ~~ 'nonaq# toidi%':text)	
4	Planning Time: 0.146 ms	
5	JIT:	
6	[...] Functions: 4	
7	[...] Options: Inlining true, Optimization true, Expressions true, Deforming true	
8	[...] Timing: Generation 1.159 ms, Inlining 0.000 ms, Optimization 0.000 ms, Emission 0.000 ms, Total 1.159 ms	
9	Execution Time: 6.837 ms	

Tak predsa sa použil. Nechcel som veriť, že to takto nebude fungovať. Ak mám popísať funkcie, lower zabezpečí, že nezáleží na case sensitive. Reverse otočí poradie, teda sa dá pomocou LIKE dobre vyhľadávať v btree ako som popísal vyššie.

## Úloha č.14

### EXPLAIN ANALYZE

```
SELECT * FROM accounts
```

```
WHERE followers_count < 10
```

```
AND friends_count > 1000
```

```
ORDER BY statuses_count DESC
```

	QUERY PLAN	
	text	
1	Gather Merge (cost=239305.26..248807.72 rows=81444 width=119) (actual time=558.653..561.973 rows=719 loops=1)	
2	[...] Workers Planned: 2	
3	[...] Workers Launched: 2	
4	[...] -> Sort (cost=238305.24..238407.04 rows=40722 width=119) (actual time=532.879..532.914 rows=240 loops=3)	
5	[...] Sort Key: statuses_count DESC	
6	[...] Sort Method: quicksort Memory: 68kB	
7	[...] Worker 0: Sort Method: quicksort Memory: 62kB	
8	[...] Worker 1: Sort Method: quicksort Memory: 67kB	
9	[...] -> Parallel Seq Scan on accounts (cost=0.00..232681.25 rows=40722 width=119) (actual time=10.029..532.310 rows=240 loops=3)	
10	[...] Filter: ((followers_count < 10) AND (friends_count > 1000))	
11	[...] Rows Removed by Filter: 3228840	
12	Planning Time: 0.140 ms	
13	JIT:	
14	[...] Functions: 6	
15	[...] Options: Inlining false, Optimization false, Expressions true, Deforming true	
16	[...] Timing: Generation 1.524 ms, Inlining 0.000 ms, Optimization 1.059 ms, Emission 13.168 ms, Total 15.752 ms	
17	Execution Time: 562.613 ms	

```
CREATE INDEX idx_followers_count ON accounts USING btree (followers_count);
```

```
CREATE INDEX idx_friends_count ON accounts USING btree (friends_count);
```

```
CREATE INDEX idx_statuses_count ON accounts USING btree (statuses_count DESC);
```

	QUERY PLAN	
	text	
1	Gather Merge (cost=221282.61..230898.72 rows=82418 width=119) (actual time=742.133..750.110 rows=719 loops=1)	
2	[...] Workers Planned: 2	
3	[...] Workers Launched: 2	
4	[...] -> Sort (cost=220282.59..220385.61 rows=41209 width=119) (actual time=712.629..712.657 rows=240 loops=3)	
5	[...] Sort Key: statuses_count DESC	
6	[...] Sort Method: quicksort Memory: 70kB	
7	[...] Worker 0: Sort Method: quicksort Memory: 61kB	
8	[...] Worker 1: Sort Method: quicksort Memory: 66kB	
9	[...] -> Parallel Bitmap Heap Scan on accounts (cost=26265.00..214586.28 rows=41209 width=119) (actual time=176.157..712.012 rows=24...	
10	[...] Recheck Cond: ((followers_count < 10) AND (friends_count > 1000))	
11	[...] Rows Removed by Index Recheck: 1917708	
12	[...] Heap Blocks: exact=15765 lossy=32581	
13	[...] -> BitmapAnd (cost=26265.00..26265.00 rows=98902 width=0) (actual time=185.031..185.032 rows=0 loops=1)	
14	[...] -> Bitmap Index Scan on idx_followers_count (cost=0.00..5604.99 rows=509407 width=0) (actual time=67.984..67.984 rows=511595 loo...	
15	[...] Index Cond: (followers_count < 10)	
16	[...] -> Bitmap Index Scan on idx_friends_count (cost=0.00..20610.31 rows=1880784 width=0) (actual time=111.512..111.512 rows=184611...	
17	[...] Index Cond: (friends_count > 1000)	
18	Planning Time: 0.538 ms	
19	JIT:	
20	[...] Functions: 6	
21	[...] Options: Inlining false, Optimization false, Expressions true, Deforming true	
22	[...] Timing: Generation 1.670 ms, Inlining 0.000 ms, Optimization 1.295 ms, Emission 13.914 ms, Total 16.879 ms	
23	Execution Time: 750.829 ms	

CREATE INDEX idx\_acc\_fol\_fr\_stat ON accounts USING btree (statuses\_count DESC, followers\_count, friends\_count); - nic spravilo

CREATE INDEX CONCURRENTLY idx\_statuses\_count ON accounts (statuses\_count DESC NULLS LAST)  
- Nic nespravilo

Žiadny z indexov vyššie a ani kombinacia nepomohla požadovanému selectu. Je to zapríčinené tým, že index má zrejme väčšiu réžiu ako klasické sekvenčné čítanie. Hľadal som aj spôsoby ako urchýliť order by, čo trvá suverénne najdlhšie ale bez úspechu ako je možné vidieť vyššie.



## Úloha č.15

```
CREATE INDEX CONCURRENTLY idx_acc  
  
ON accounts (statuses_count DESC)  
  
WHERE followers_count < 10  
  
AND friends_count > 1000;
```

	QUERY PLAN	
	text	
1	Gather Merge (cost=194040.56..203543.03 rows=81444 width=119) (actual time=39.937..43.042 rows=719 loops=1)	
2	[...] Workers Planned: 2	
3	[...] Workers Launched: 2	
4	[...] -> Sort (cost=193040.54..193142.35 rows=40722 width=119) (actual time=1.495..1.535 rows=240 loops=3)	
5	[...] Sort Key: statuses_count DESC	
6	[...] Sort Method: quicksort Memory: 147kB	
7	[...] Worker 0: Sort Method: quicksort Memory: 25kB	
8	[...] Worker 1: Sort Method: quicksort Memory: 25kB	
9	[...] -> Parallel Bitmap Heap Scan on accounts (cost=44.30..187416.55 rows=40722 width=119) (actual time=0.303..1.319 rows=240 loops=...	
10	[...] Recheck Cond: ((followers_count < 10) AND (friends_count > 1000))	
11	[...] Heap Blocks: exact=718	
12	[...] -> Bitmap Index Scan on idx_acc (cost=0.00..19.87 rows=97733 width=0) (actual time=0.291..0.292 rows=719 loops=1)	
13	Planning Time: 0.259 ms	
14	JIT:	
15	[...] Functions: 6	
16	[...] Options: Inlining false, Optimization false, Expressions true, Deforming true	
17	[...] Timing: Generation 1.960 ms, Inlining 0.000 ms, Optimization 0.000 ms, Emission 0.000 ms, Total 1.960 ms	
18	Execution Time: 43.778 ms	

Po vytvorení takto konkrétneho indexu a použitia pôvodného selectu sa planer rozhodol ho použiť a výrazne to urýchlilo tento select. Je to zrejme práve tým, že sú v indexe len práve tie hodnoty, ktoré hľadám v selecte, čiže je to predpripravené.

Po kontrolovaní mi došlo, že toto zložený index nie je a preto som tam nejaký prvý nastrel hodil takto

```
CREATE INDEX idx_acc ON accounts (followers_count, friends_count)
```

```
CREATE INDEX
```

```
Query returned successfully in 9 secs 571 msec.
```

EXPLAIN ANALYZE

SELECT \* FROM accounts

WHERE followers\_count < 10

AND friends\_count > 1000

ORDER BY statuses\_count DESC

QUERY PLAN		
	text	
1	Gather Merge (cost=203312.84..212928.95 rows=82418 width=119) (actual time=251.865..256.676 rows=719 loops=1)	
2	[...] Workers Planned: 2	
3	[...] Workers Launched: 2	
4	[...] -> Sort (cost=202312.82..202415.84 rows=41209 width=119) (actual time=190.939..190.967 rows=240 loops=3)	
5	[...] Sort Key: statuses_count DESC	
6	[...] Sort Method: quicksort Memory: 58kB	
7	[...] Worker 0: Sort Method: quicksort Memory: 71kB	
8	[...] Worker 1: Sort Method: quicksort Memory: 67kB	
9	[...] -> Parallel Bitmap Heap Scan on accounts (cost=8295.23..196616.51 rows=41209 width=119) (actual time=115.820..189.9...	
10	[...] Recheck Cond: (((followers_count < 10) AND (friends_count > 1000)))	
11	[...] Heap Blocks: exact=215	
12	[...] -> Bitmap Index Scan on idx_acc (cost=0.00..8270.50 rows=98902 width=0) (actual time=175.149..175.150 rows=719 loop...	
13	[...] Index Cond: (((followers_count < 10) AND (friends_count > 1000)))	
14	Planning Time: 1.654 ms	
15	JIT:	
16	[...] Functions: 6	
17	[...] Options: Inlining false, Optimization false, Expressions true, Deforming true	
18	[...] Timing: Generation 9.868 ms, Inlining 0.000 ms, Optimization 0.000 ms, Emission 0.000 ms, Total 9.868 ms	
19	Execution Time: 257.844 ms	

Vyzerá, že pomohlo a skrátil sa tak čas. Keďže sú indexy prepojené, tak nájdenie správnych pre jeden zmenší množinu pre druhý a tým padom je celková množina hľadania ešte menšia ako pri separátnych indexoch. Menej pamäte = rýchlejšie – menšia réžia.

## Úloha č.16

EXPLAIN ANALYZE

SELECT \* FROM accounts

WHERE followers\_count < 1000

AND friends\_count > 1000

ORDER BY statuses\_count DESC

QUERY PLAN		
text		
1	Gather Merge (cost=357208.11..489071.18 rows=1130176 width=119) (actual time=794.988..1123.652 rows=740655 loops=1)	
2	[...] Workers Planned: 2	
3	[...] Workers Launched: 2	
4	[...] -> Sort (cost=356208.09..357620.81 rows=565088 width=119) (actual time=762.880..847.489 rows=246885 loops=3)	
5	[...] Sort Key: statuses_count DESC	
6	[...] Sort Method: external merge Disk: 38056kB	
7	[...] Worker 0: Sort Method: external merge Disk: 34864kB	
8	[...] Worker 1: Sort Method: external merge Disk: 35456kB	
9	[...] -> Parallel Seq Scan on accounts (cost=0.00..232681.25 rows=565088 width=119) (actual time=5.940..591.758 rows=246885 loops=3)	
10	[...] Filter: ((followers_count < 1000) AND (friends_count > 1000))	
11	[...] Rows Removed by Filter: 2982195	
12	Planning Time: 0.489 ms	
13	JIT:	
14	[...] Functions: 6	
15	[...] Options: Inlining false, Optimization false, Expressions true, Deforming true	
16	[...] Timing: Generation 1.709 ms, Inlining 0.000 ms, Optimization 1.146 ms, Emission 16.013 ms, Total 18.868 ms	
17	Execution Time: 1159.305 ms	

Tento zložený index sa zrejme nepoužije, pretože vybraných záznamov je veľa. Teda by index trval dlhšie ako sekvenčný sken.

## Úloha č.17

### Vyhľadanie bez indexu

EXPLAIN ANALYSE SELECT screen\_name, name FROM accounts WHERE screen\_name = 'realDonaldTrump'

	QUERY PLAN	
	text	
1	Gather (cost=1000.00..223590.48 rows=1 width=26) (actual time=13.925..758.460 rows=1 loops=1)	
2	[...] Workers Planned: 2	
3	[...] Workers Launched: 2	
4	[...] -> Parallel Seq Scan on accounts (cost=0.00..222590.38 rows=1 width=26) (actual time=451.424..698.020 rows=0 loops=3)	
5	[...] Filter: ((screen_name)::text = 'realDonaldTrump'::text)	
6	[...] Rows Removed by Filter: 3229080	
7	Planning Time: 0.152 ms	
8	JIT:	
9	[...] Functions: 12	
10	[...] Options: Inlining false, Optimization false, Expressions true, Deforming true	
11	[...] Timing: Generation 2.976 ms, Inlining 0.000 ms, Optimization 1.726 ms, Emission 25.898 ms, Total 30.600 ms	
12	Execution Time: 759.438 ms	

### Vytvorenie indexu

CREATE INDEX idx\_screen\_name\_btree ON accounts USING btree (screen\_name);

```
CREATE INDEX
```

```
Query returned successfully in 51 secs 757 msec.
```

CREATE EXTENSION pg\_trgm;


CREATE INDEX idx\_screen\_name\_gin ON accounts USING gin(screen\_name gin\_trgm\_ops);

```
CREATE INDEX
```


```
Query returned successfully in 1 min 46 secs.
```

### Vyhľadanie s indexom

EXPLAIN ANALYSE SELECT screen\_name, name FROM accounts WHERE screen\_name = 'realDonaldTrump'

QUERY PLAN		
	text	
1	Index Scan using idx_screen_name_btree on accounts (cost=0.43..8.45 rows=1 width=26) (actual time=0.087..0.089 rows=1 lo...	
2	[...] Index Cond: ((screen_name)::text = 'realDonaldTrump'::text)	
3	Planning Time: 0.480 ms	
4	Execution Time: 0.117 ms	

EXPLAIN ANALYSE SELECT screen\_name, name FROM accounts WHERE screen\_name LIKE '%ldonaldt%'

QUERY PLAN		
	text	
1	Bitmap Heap Scan on accounts (cost=127.51..3790.19 rows=969 width=26) (actual time=4.633..4.860 rows=3 loops=1)	
2	[...] Recheck Cond: ((screen_name)::text ~~ '%ldonaldt%'::text)	
3	[...] Rows Removed by Index Recheck: 21	
4	[...] Heap Blocks: exact=24	
5	[...] -> Bitmap Index Scan on idx_screen_name_gin (cost=0.00..127.27 rows=969 width=0) (actual time=4.527..4.528 rows=24 l...	
6	[...] Index Cond: ((screen_name)::text ~~ '%ldonaldt%'::text)	
7	Planning Time: 0.476 ms	
8	Execution Time: 4.934 ms	

Btree index funguje lepšie pre presný výraz, pretože z názvu je to strom, ktorý traverzuje podľa stringu a teda je to logaritmicky rýchlo.

Pri hľadaní ldonaldt už sa btree nedá využiť, pretože nemá podľa čoho traverzovať strom a vyberie bitmap heap scan z GIN indexu, ktorý je na takýto prípad robený.

## Úloha č.18

### Prvý explain prvého selectu

```
EXPLAIN SELECT * FROM tweets t
```


```
JOIN accounts a ON a.id = t.author_id
```

```
WHERE to_tsvector('english', t.content) @@ to_tsquery('english', 'John & Oliver')
```

```
OR to_tsvector('english', a.description) @@ to_tsquery('english', 'John & Oliver')
```

```
OR to_tsvector('english', a.name) @@ to_tsquery('english', 'John & Oliver')
```

```
ORDER BY t.retweet_count DESC
```

QUERY PLAN		
	text	
1	Gather Merge (cost=2746085.02..2746318.13 rows=1998 width=431)	
2	[...] Workers Planned: 2	
3	[...] -> Sort (cost=2745084.99..2745087.49 rows=999 width=431)	
4	[...] Sort Key: t.retweet_count DESC	
5	[...] -> Parallel Hash Join (cost=333905.88..2745035.22 rows=999 width=431)	
6	[...] Hash Cond: (t.author_id = a.id)	
7	[...] Join Filter: ((to_tsvector('english':regconfig, t.content) @@ "john" & "oliv":tsquery) OR (to_tsvector('english':regconfig, a.desc...	
8	[...] -> Parallel Seq Scan on tweets t (cost=0.00..1212271.54 rows=13323354 width=312)	
9	[...] -> Parallel Hash (cost=212499.50..212499.50 rows=4036350 width=119)	
10	[...] -> Parallel Seq Scan on accounts a (cost=0.00..212499.50 rows=4036350 width=119)	
11	JIT:	
12	[...] Functions: 13	
13	[...] Options: Inlining true, Optimization true, Expressions true, Deforming true	

```
CREATE INDEX idx_content ON tweets USING gin(to_tsvector('english', content))
```

```
CREATE INDEX idx_description ON accounts USING gin(to_tsvector('english', description))
```

```
CREATE INDEX idx_name ON accounts USING gin(to_tsvector('english', name))
```

```
CREATE INDEX
```

```
Query returned successfully in 37 min 25 secs.
```

### Vyhľadanie s indexami

	QUERY PLAN	
	text	
1	Gather Merge (cost=2746085.02..2746318.13 rows=1998 width=431)	
2	[...] Workers Planned: 2	
3	[...] -> Sort (cost=2745084.99..2745087.49 rows=999 width=431)	
4	[...] Sort Key: t.retweet_count DESC	
5	[...] -> Parallel Hash Join (cost=333905.88..2745035.22 rows=999 width=431)	
6	[...] Hash Cond: (t.author_id = a.id)	
7	[...] Join Filter: ((to_tsvector('english':regconfig, t.content) @@ "john" & "oliv":tsquery) OR (to_tsvector('english':regconfig, a.desc...	
8	[...] -> Parallel Seq Scan on tweets t (cost=0.00..1212271.54 rows=13323354 width=312)	
9	[...] -> Parallel Hash (cost=212499.50..212499.50 rows=4036350 width=119)	
10	[...] -> Parallel Seq Scan on accounts a (cost=0.00..212499.50 rows=4036350 width=119)	
11	JIT:	
12	[...] Functions: 13	
13	[...] Options: Inlining true, Optimization true, Expressions true, Deforming true	

Vidno, že sa nič nezmenilo. Googlil som a vyzerá to tak, že planer nemá rád OR a teda je potrebné to nejak prepísať.

Našiel som, že existuje v postgrese UNION, ktorý spojí záznamy zo všetkých nad sebou selectov. Pomocou tohto viem rozdeliť všetky 3 selecty a spojiť z nich výsledky. Rozmýšľal som, že by bolo možné rozdeliť tieto selecty aj bez JOIN ale bolo by tam komplikovanejšie spojiť záznamy na konci.

EXPLAIN ANALYZE

SELECT \* FROM tweets t

JOIN accounts a ON a.id = t.author\_id

WHERE to\_tsvector('english', t.content) @@ to\_tsquery('english', 'John & Oliver')

UNION

SELECT \* FROM tweets t

JOIN accounts a ON a.id = t.author\_id

WHERE to\_tsvector('english', a.description) @@ to\_tsquery('english', 'John & Oliver')

UNION

SELECT \* FROM tweets t

JOIN accounts a ON a.id = t.author\_id

WHERE to\_tsvector('english', a.name) @@ to\_tsquery('english', 'John & Oliver')

	QUERY PLAN	
	text	
1	HashAggregate (cost=120860.47..120884.44 rows=2397 width=1152) (actual time=48.534..48.893 rows=612 loops=1)	
2	[...] Group Key: t.id, t.content, t.location, t.retweet_count, t.favorite_count, t.happened_at, t.author_id, t.country_id, t.parent_id, t.po...	
3	[...] Batches: 1 Memory Usage: 601kB	
4	[...] -> Append (cost=58.63..120740.62 rows=2397 width=1152) (actual time=43.803..47.829 rows=612 loops=1)	
5	[...] -> Nested Loop (cost=58.63..10104.25 rows=799 width=431) (actual time=43.802..46.552 rows=457 loops=1)	
6	[...] -> Bitmap Heap Scan on tweets t (cost=58.20..3398.71 rows=799 width=312) (actual time=1.095..1.587 rows=457 loops=1)	
7	[...] Recheck Cond: (to_tsvector('english'::regconfig, content) @@ "john" & "oliv"::tsquery)	
8	[...] Heap Blocks: exact=398	
9	[...] -> Bitmap Index Scan on idx_content (cost=0.00..58.00 rows=799 width=0) (actual time=1.056..1.056 rows=457 loops=1)	
10	[...] Index Cond: (to_tsvector('english'::regconfig, content) @@ "john" & "oliv"::tsquery)	
11	[...] -> Index Scan using idx_id on accounts a (cost=0.43..8.39 rows=1 width=119) (actual time=0.004..0.004 rows=1 loops=457)	
12	[...] Index Cond: (id = t.author_id)	
13	[...] -> Nested Loop (cost=45.88..55300.21 rows=799 width=431) (actual time=0.276..0.527 rows=131 loops=1)	
14	[...] -> Bitmap Heap Scan on accounts a_1 (cost=45.88..1050.18 rows=242 width=119) (actual time=0.245..0.276 rows=27 loop...	
15	[...] Recheck Cond: (to_tsvector('english'::regconfig, description) @@ "john" & "oliv"::tsquery)	
16	[...] Heap Blocks: exact=27	
17	[...] -> Bitmap Index Scan on idx_description (cost=0.00..45.82 rows=242 width=0) (actual time=0.239..0.239 rows=27 loops=1)	
18	[...] Index Cond: (to_tsvector('english'::regconfig, description) @@ "john" & "oliv"::tsquery)	
19	[...] -> Index Scan using idx_author_id on tweets t_1 (cost=0.00..223.62 rows=55 width=312) (actual time=0.003..0.007 rows=5 l...	
20	[...] Index Cond: (author_id = a_1.id)	
21	[...] -> Nested Loop (cost=45.88..55300.21 rows=799 width=431) (actual time=0.610..0.706 rows=24 loops=1)	
22	[...] -> Bitmap Heap Scan on accounts a_2 (cost=45.88..1050.18 rows=242 width=119) (actual time=0.594..0.613 rows=18 loop...	
23	[...] Recheck Cond: (to_tsvector('english'::regconfig, (name)::text) @@ "john" & "oliv"::tsquery)	
24	[...] Heap Blocks: exact=18	
24	[...] Heap Blocks: exact=18	
25	[...] -> Bitmap Index Scan on idx_name (cost=0.00..45.82 rows=242 width=0) (actual time=0.590..0.590 rows=18 loops=1)	
26	[...] Index Cond: (to_tsvector('english'::regconfig, (name)::text) @@ "john" & "oliv"::tsquery)	
27	[...] -> Index Scan using idx_author_id on tweets t_2 (cost=0.00..223.62 rows=55 width=312) (actual time=0.003..0.004 rows=1 l...	
28	[...] Index Cond: (author_id = a_2.id)	
29	Planning Time: 0.879 ms	
30	JIT:	
31	[...] Functions: 29	
32	[...] Options: Inlining false, Optimization false, Expressions true, Deforming true	
33	[...] Timing: Generation 5.918 ms, Inlining 0.000 ms, Optimization 2.282 ms, Emission 39.305 ms, Total 47.505 ms	
34	Execution Time: 55.140 ms	

Teraz už len zoradiť pomocou ORDER BY záznamy, ktoré mi to vyhľadalo

Tu je zasa problém, že sem nejde len tak napísať ORDER BY.

Na toto mi napadlo riešenie, že to skúsím dať ako subselecty a potom zoradiť v main selecte.



Dal som to celé do subselectu a ide to pekne.

EXPLAIN ANALYZE

```
SELECT * FROM (SELECT * FROM tweets t
```

```
JOIN accounts a ON a.id = t.author_id
```

```
WHERE to_tsvector('english', t.content) @@ to_tsquery('english', 'John & Oliver')
```

```
UNION
```

```
SELECT * FROM tweets t
```

```
JOIN accounts a ON a.id = t.author_id
```

```
WHERE to_tsvector('english', a.description) @@ to_tsquery('english', 'John & Oliver')
```

```
UNION
```

```
SELECT * FROM tweets t
```

```
JOIN accounts a ON a.id = t.author_id
```

```
WHERE to_tsvector('english', a.name) @@ to_tsquery('english', 'John & Oliver')) u
```

```
ORDER BY u.retweet_count DESC
```

1	Sort (cost=121042.97..121048.96 rows=2397 width=1152) (actual time=53.732..53.760 rows=612 loops=1)
2	[...] Sort Key: t.retweet_count DESC
3	[...] Sort Method: quicksort Memory: 335kB
4	[...] -> HashAggregate (cost=120860.47..120884.44 rows=2397 width=1152) (actual time=53.028..53.439 rows=612 loops=1)
5	[...] Group Key: t.id, t.content, t.location, t.retweet_count, t.favorite_count, t.happened_at, t.author_id, t.country_id, t.parent_id, t.po...
6	[...] Batches: 1 Memory Usage: 601kB
7	[...] -> Append (cost=58.63..120740.62 rows=2397 width=1152) (actual time=48.097..52.114 rows=612 loops=1)
8	[...] -> Nested Loop (cost=58.63..10104.25 rows=799 width=431) (actual time=48.095..50.920 rows=457 loops=1)
9	[...] -> Bitmap Heap Scan on tweets t (cost=58.20..3398.71 rows=799 width=312) (actual time=1.189..1.700 rows=457 loops=1)
10	[...] Recheck Cond: (to_tsvector('english':regconfig, content) @@ "john" & "oliv":tsquery)
11	[...] Heap Blocks: exact=398
12	[...] -> Bitmap Index Scan on idx_content (cost=0.00..58.00 rows=799 width=0) (actual time=1.130..1.130 rows=457 loops=1)
13	[...] Index Cond: (to_tsvector('english':regconfig, content) @@ "john" & "oliv":tsquery)
14	[...] -> Index Scan using idx_id on accounts a (cost=0.43..8.39 rows=1 width=119) (actual time=0.004..0.004 rows=1 loops=457)
15	[...] Index Cond: (id = t.author_id)
16	[...] -> Nested Loop (cost=45.88..55300.21 rows=799 width=431) (actual time=0.227..0.489 rows=131 loops=1)
17	[...] -> Bitmap Heap Scan on accounts a_1 (cost=45.88..1050.18 rows=242 width=119) (actual time=0.204..0.234 rows=27 loop...
18	[...] Recheck Cond: (to_tsvector('english':regconfig, description) @@ "john" & "oliv":tsquery)
19	[...] Heap Blocks: exact=27
20	[...] -> Bitmap Index Scan on idx_description (cost=0.00..45.82 rows=242 width=0) (actual time=0.199..0.199 rows=27 loops=1)
21	[...] Index Cond: (to_tsvector('english':regconfig, description) @@ "john" & "oliv":tsquery)
22	[...] -> Index Scan using idx_author_id on tweets t_1 (cost=0.00..223.62 rows=55 width=312) (actual time=0.003..0.008 rows=5 l...
23	[...] Index Cond: (author_id = a_1.id)
24	[...] -> Nested Loop (cost=45.88..55300.21 rows=799 width=431) (actual time=0.568..0.660 rows=24 loops=1)
24	[...] -> Nested Loop (cost=45.88..55300.21 rows=799 width=431) (actual time=0.568..0.660 rows=24 loops=1)
25	[...] -> Bitmap Heap Scan on accounts a_2 (cost=45.88..1050.18 rows=242 width=119) (actual time=0.556..0.575 rows=18 loop...
26	[...] Recheck Cond: (to_tsvector('english':regconfig, (name)::text) @@ "john" & "oliv":tsquery)
27	[...] Heap Blocks: exact=18
28	[...] -> Bitmap Index Scan on idx_name (cost=0.00..45.82 rows=242 width=0) (actual time=0.552..0.552 rows=18 loops=1)
29	[...] Index Cond: (to_tsvector('english':regconfig, (name)::text) @@ "john" & "oliv":tsquery)
30	[...] -> Index Scan using idx_author_id on tweets t_2 (cost=0.00..223.62 rows=55 width=312) (actual time=0.003..0.004 rows=1 l...
31	[...] Index Cond: (author_id = a_2.id)
32	Planning Time: 0.911 ms
33	JIT:
34	[...] Functions: 29
35	[...] Options: Inlining false, Optimization false, Expressions true, Deforming true
36	[...] Timing: Generation 7.290 ms, Inlining 0.000 ms, Optimization 2.761 ms, Emission 43.200 ms, Total 53.252 ms
37	Execution Time: 61.468 ms

