PDT - part 2

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?

QUERY PLAN
Gather (cost=1000.00..223590.46 rows=1 width=11) (actual time=38.442..1657.232 rows=1 loops=1)
Workers Planned: 2
Workers Launched: 2
-> Parallel Seq Scan on accounts (cost=0.00..222590.36 rows=1 width=11) (actual time=996.475..1518.365 rows=0 loops=3)
Filter: ((screen_name)::text = 'realDonaldTrump'::text)
Rows Removed by Filter: 3229079
Planning Time: 0.109 ms
JIT:
Functions: 12
Options: Inlining false, Optimization false, Expressions true, Deforming true
Timing: Generation 2.250 ms, Inlining 0.000 ms, Optimization 0.945 ms, Emission 16.015 ms, Total 19.209 ms
Execution Time: 1659.895 ms

Vybral se paralel sequence scan. Neexistuje zatím žádný index a chceme match na jednu konkrétní hodnotu, Postgres musí tedy projít všechny hodnoty po jedné.

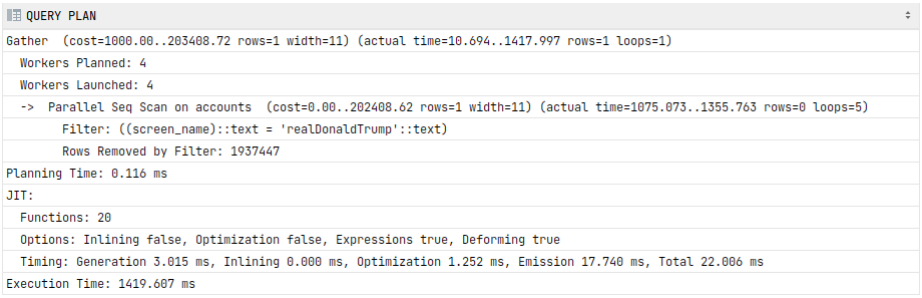
Paralel znamená, že na dané query pracovalo více “procesů”.

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?

QUERY PLAN
Gather (cost=1000.00..223590.46 rows=1 width=11) (actual time=5.361..1566.133 rows=1 loops=1)
Workers Planned: 2
Workers Launched: 2
-> Parallel Seq Scan on accounts (cost=0.00..222590.36 rows=1 width=11) (actual time=1011.652..1531.028 rows=0 loops=3)
Filter: ((screen_name)::text = 'realDonaldTrump'::text)
Rows Removed by Filter: 3229079
Planning Time: 0.061 ms
JIT:
Functions: 12
Options: Inlining false, Optimization false, Expressions true, Deforming true
Timing: Generation 1.522 ms, Inlining 0.000 ms, Optimization 0.703 ms, Emission 8.806 ms, Total 11.031 ms
Execution Time: 1566.959 ms

Na selektě pracovali defaultně 2 workry. Díky nim lze queries provádět paralelně.



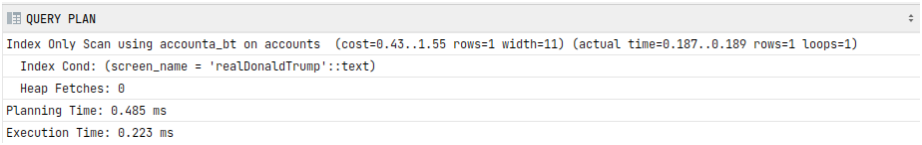
```
QUERY PLAN
Gather  (cost=1000.00..203408.72 rows=1 width=11) (actual time=10.694..1417.997 rows=1 loops=1)
  Workers Planned: 4
  Workers Launched: 4
  ->  Parallel Seq Scan on accounts  (cost=0.00..202408.62 rows=1 width=11) (actual time=1075.073..1355.763 rows=0 loops=5)
        Filter: ((screen_name)::text = 'realDonaldTrump'::text)
        Rows Removed by Filter: 1937447
Planning Time: 0.116 ms
JIT:
  Functions: 20
  Options: Inlining false, Optimization false, Expressions true, Deforming true
  Timing: Generation 3.015 ms, Inlining 0.000 ms, Optimization 1.252 ms, Emission 17.740 ms, Total 22.006 ms
Execution Time: 1419.607 ms
```

Při zdvihnutí počtu workerů se snížil čas. Velká úspora je při změně 0 workerů na 2 (o 31%). Při 4 workerech není úspora o moc větší. Oproti 0 workerů se čas snížil o 36%. S procesy je notně spjatá i další režije.

Maximální počet workerů na úlohu se mění pomocí *max_parallel_workers_per_gather*. Strop je pak nastaven proměnnou *max_parallel_workers*.

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?

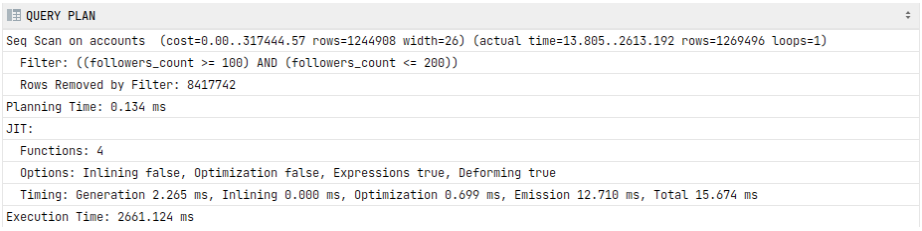


```
QUERY PLAN
Index Only Scan using accounta_bt on accounts  (cost=0.43..1.55 rows=1 width=11) (actual time=0.187..0.189 rows=1 loops=1)
  Index Cond: (screen_name = 'realDonaldTrump'::text)
  Heap Fetches: 0
Planning Time: 0.485 ms
Execution Time: 0.223 ms
```

Čas se snížil 7439x oproti query v části 1. Nebylo potřeba procházet všechny záznamy. Existují už indexy v rámci btree, které se využily.

4.

Vyberte používateľov, ktorý 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?



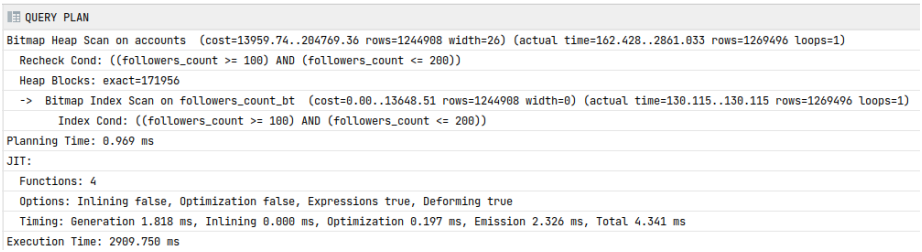
```
QUERY PLAN
Seq Scan on accounts  (cost=0.00..317444.57 rows=1244908 width=26) (actual time=13.805..2613.192 rows=1269496 loops=1)
  Filter: ((followers_count >= 100) AND (followers_count <= 200))
  Rows Removed by Filter: 8417742
Planning Time: 0.134 ms
JIT:
  Functions: 4
  Options: Inlining false, Optimization false, Expressions true, Deforming true
  Timing: Generation 2.265 ms, Inlining 0.000 ms, Optimization 0.699 ms, Emission 12.710 ms, Total 15.674 ms
Execution Time: 2661.124 ms
```

Chování je podobné první úloze. Tentokrát se ale jedná o sekvenční scan bez paralelismu. Prošly se všechny řádky, které se vyfiltrovali.

Jinou možnost moc nemá, jelikož zatím není vytvořený index.

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?

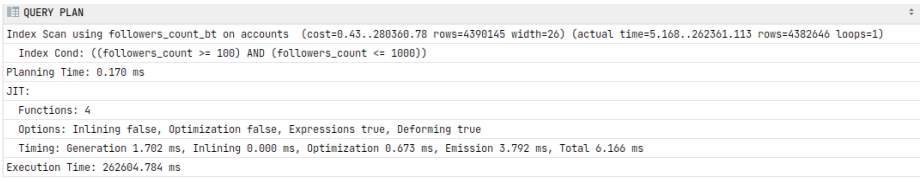


```
QUERY PLAN
Bitmap Heap Scan on accounts  (cost=13959.74..204769.36 rows=1244908 width=26) (actual time=162.428..2861.033 rows=1269496 loops=1)
  Recheck Cond: ((followers_count >= 100) AND (followers_count <= 200))
  Heap Blocks: exact=171956
  ->  Bitmap Index Scan on followers_count_bt  (cost=0.00..13648.51 rows=1244908 width=0) (actual time=130.115..130.115 rows=1269496 loops=1)
        Index Cond: ((followers_count >= 100) AND (followers_count <= 200))
Planning Time: 0.969 ms
JIT:
  Functions: 4
  Options: Inlining false, Optimization false, Expressions true, Deforming true
  Timing: Generation 1.818 ms, Inlining 0.000 ms, Optimization 0.197 ms, Emission 2.326 ms, Total 4.341 ms
Execution Time: 2909.750 ms
```

Nejřív se vytvoří Bitmap index pomocí projití tabulky scanem, přičemž bitmapy se vytváří na základě podmínky. Tyto bitmapy se sloučí pomocí AND. Každá bitmapa je k jedné stránce. Nakonec se udělá scan po stránkách, přičemž podle bitmapy dané stránky se pozná, jestli má smysl ji navštívit. Ne všechny tuples ve stránce odpovídají podmínce. Musí být znovu zkontrolovány, proto recheck condition.

6.

Vyberte používateľov, ktorí majú followers_count väčší, rovný ako 100 a zároveň menší, rovný 1000? V čom je rozdiel, prečo?



```
QUERY PLAN
Index Scan using followers_count_bt on accounts  (cost=0.43..280360.78 rows=4390145 width=26) (actual time=5.168..262361.113 rows=4382646 loops=1)
  Index Cond: ((followers_count >= 100) AND (followers_count <= 1000))
Planning Time: 0.170 ms
JIT:
  Functions: 4
  Options: Inlining false, Optimization false, Expressions true, Deforming true
  Timing: Generation 1.702 ms, Inlining 0.000 ms, Optimization 0.673 ms, Emission 3.792 ms, Total 6.166 ms
Execution Time: 262604.784 ms
```

Počet výsledků je polovina ze všech hodnot. Bitmat index scan zdřejmě již neměl význam, tak se použil idex scan.

Rozsah	Počet výsledků	Počet unikátních hodnot
<100, 200>	1 269 496	101
<100, 10000>	4 382 646	901
all	8 786 922	83 576

7.

Vytvorte daľšie 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?

Bez indexů: 0.141 ms S indexy: 308.905 ms

Indexy sice urychlují SELECT, ale při vkládání se musí počítat s dodatečnou režií. V případě btree se dost pravděpodobně budou rozdělovat, nebo jinak přeskládávat uzly, tak aby strom nedegradoval.

Navíc se musí přepočítat více indexů.

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?

Sloupec	Délka vytváření indexu	Počet unikátních hodnot	Velikost indexu
retwee_count	26 s 602 ms	23880	214 MB
contend	4 m 42 s 911 ms	14723540	3136 MB

Je ovlivněna počtem unikátních hodnot. Čím větší, tím větší musí být i vytvořený strom a tím déle to trvá.

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');

name	magic	version	root	level	fastroot	fastlevel	oldest_xact	last_cleanup_num_tuples	allequalimage
accounts_screen_name_bt	340322	4	222	2	222	2	0	-1	true
followers_count_bt	340322	4	209	2	209	2	0	-1	true
accounts_name_bt	340322	4	23641	3	23641	3	0	-1	true
accounts_friends_count_bt	340322	4	210	2	210	2	0	-1	true
accounts_description_bt	340322	4	54822	4	54822	4	0	-1	true
tweets_retweet_count_bt	340322	4	209	2	209	2	0	-1	true
tweets_content_bt	340322	4	175412	5	175412	5	0	-1	true

- magic: no idea
- version: verze btree
- root: umístění kořene
- level: level kořene od spodu
- oldest_xact: TransactionId/xid field
- last_clean_up_number_tuples: ??
- allequalimage: ??

Fast root

Při delete se nikdy nemaže nejpravější stránka, nejde tedy snížit výšku stromu. Při velkých promazání může z toho důvodu být strom velmi úzký - o šířce např. 1 stránky, proto se drží záznam o nejnižším jednostránkovém levelu. Při vyhledávání se nemusí procházet od kořene dolů, ale začne se až na tomto nižším "kořeni".

V našem případě se žádné masivní delety nekonaly, proto root level a fastroot level jsou stejné. Největším indexem je tweets_content_bt, který také obsahuje nejvíce hodnot, proto má největší level.

c. select type, live_items, dead_items, avg_item_size, page_size, free_size from bt_page_stats('idx_content',1000);

name	type	live_items	dead_items	avg_item_size	page_size	free_size
accounts_screen_name_bt	l	271	0	23	8192	816
followers_count_bt	l	10	0	729	8192	812
accounts_name_bt	l	220	0	29	8192	796
accounts_friends_count_bt	l	10	0	729	8192	812
accounts_description_bt	l	10	0	729	8192	812
tweets_retweet_count_bt	l	10	0	729	8192	812
tweets_content_bt	l	37	0	199	8192	632

Dostáváme informace o single pages. Z předchozích informací je jasné, že jediná taková stránka je root.

- type:
- live_items: použitelné položkx
- dead_items: ještě fyzicky neodstraněné položky, ale už vyřazené
- avg_item_size: velikost 1 položky
- page_size: velikost stránky
- free_size: volné místo stránky

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

Infromace o položkách na každé stránce.

- itemoffset: pořadí položky
- ctid: zavedeno kvůli duplikátním key value, díky ctid zajištěno seřazení a uložení ve správném pořadí
- itemlen: délka položky
- nulls: ??
- htids: ??
- tids: ??

10.

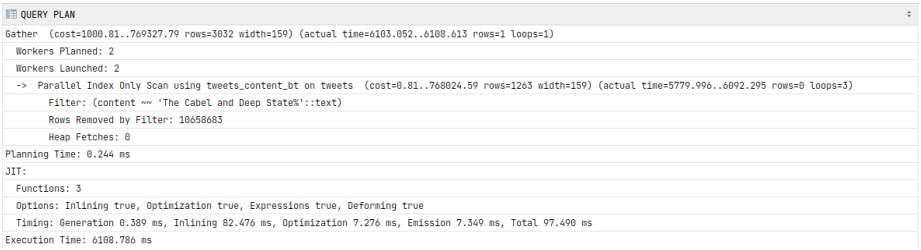
Vyhľadajte v tweets.content meno „Gates“ na ľubovoľnom mieste a porovnajte výsledok po tom, ako content naindexujete pomocou btree. V čom je rozdiel a prečo?

bez indexu: 469ms s indexem: 547ms

Žádný signifikantní rozdíl není. Btree nám nemůže pomoct. Neznáme začátek contendu, není podle čeho porovnávat, můsí se tedy projet všechny řádky scanem. V tomto případě byl použit paralel seq scan.

11.

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



```
QUERY PLAN
Gather  (cost=1000.81..769327.79 rows=3032 width=159) (actual time=6103.052..6108.613 rows=1 loops=1)
  Workers Planned: 2
  Workers Launched: 2
  ->  Parallel Index Only Scan using tweets_content_bt on tweets  (cost=0.81..768024.59 rows=1263 width=159) (actual time=5779.996..6092.295 rows=0 loops=3)
        Filter: (content ~~ 'The Cabel and Deep State%'::text)
        Rows Removed by Filter: 10658683
        Heap Fetches: 0
Planning Time: 0.244 ms
JIT:
  Functions: 3
  Options: Inlining true, Optimization true, Expressions true, Deforming true
  Timing: Generation 0.389 ms, Inlining 82.476 ms, Optimization 7.276 ms, Emission 7.349 ms, Total 97.490 ms
Execution Time: 6108.786 ms
```

V mém případě ano. Dle stránky pgdash se to ale běžně neděje.

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?

```
QUERY PLAN
Index Only Scan using tweets_content_bt on tweets  (cost=0.81..1.94 rows=3032 width=159) (actual time=1.723..1.725 rows=1 loops=1)
  Index Cond: ((content ~>=~ 'The Cabel and Deep State'::text) AND (content ~<~ 'The Cabel and Deep Statf'::text))
  Filter: (content ~~ 'The Cabel and Deep State%'::text)
  Heap Fetches: 0
Planning Time: 1.752 ms
Execution Time: 1.758 ms
```

Použití btree indexu by se mělo umožnit až přidáním parametru *text_pattern_ops*, který umožňuje indexovat přes text. Tímto spůsobem se to předpokládám muselo dělat v minulých verzích PostresSQL.

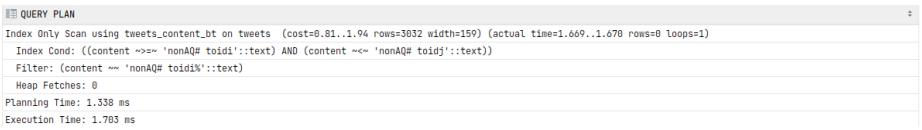
U '%Gates%' mi btree index napomůže. Tento string se nedá porovnávat.

13.

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

```
CREATE INDEX reverse_tweets_content_bt ON tweets (reverse(content) text_pattern_ops);

explain analyse select content from tweets   waiting
    where content like reverse('%idiot #QAnon');
```



```
QUERY PLAN
Index Only Scan using tweets_content_bt on tweets  (cost=0.81..1.94 rows=3032 width=159) (actual time=1.669..1.670 rows=0 loops=1)
  Index Cond: ((content ~>=~ 'nonAQ# toidi'::text) AND (content ~<~ 'nonAQ# toidj'::text))
  Filter: (content ~~ 'nonAQ# toidi%'::text)
  Heap Fetches: 0
Planning Time: 1.338 ms
Execution Time: 1.703 ms
```

1. Vytvořím reverse btree index, který seřazuje odkonce.
2. Můžu vyhledávat stringy podle konce, pokud je před porovnáním také reversnu.

14.

Nájdite účty, ktoré majú follower_count menší ako 10 a friends_count väčší ako 1000 a výsledok zoraďte podľa statuses_count. Následne spravte jednoduché indexy a popíšte ktoré má a ktoré nemá zmysel robiť a prečo.

```
QUERY PLAN
Gather Merge  (cost=236720.58..246003.69 rows=79564 width=19) (actual time=1461.168..1464.646 rows=719 loops=1)
  Workers Planned: 2
  Workers Launched: 2
  ->  Sort  (cost=235720.55..235820.01 rows=39782 width=19) (actual time=1430.175..1430.193 rows=240 loops=3)
        Sort Key: statuses_count
        Sort Method: quicksort  Memory: 42kB
        Worker 0:  Sort Method: quicksort  Memory: 37kB
        Worker 1:  Sort Method: quicksort  Memory: 38kB
        ->  Parallel Seq Scan on accounts  (cost=0.00..232681.24 rows=39782 width=19) (actual time=13.579..1429.871 rows=240 loops=3)
              Filter: ((followers_count < 10) AND (friends_count > 1000))
              Rows Removed by Filter: 3228840
Planning Time: 0.285 ms
JIT:
  Functions: 12
  Options: Inlining false, Optimization false, Expressions true, Deforming true
  Timing: Generation 2.780 ms, Inlining 0.000 ms, Optimization 1.773 ms, Emission 9.516 ms, Total 14.070 ms
Execution Time: 1466.625 ms
```

Na podmínku se použil paralel scan. Na seřazení quicksort, který se vlezl do paměti (42kb).

```
QUERY PLAN
Sort  (cost=107449.07..107690.66 rows=96637 width=19) (actual time=228.551..228.591 rows=719 loops=1)
  Sort Key: statuses_count
  Sort Method: quicksort  Memory: 69kB
  ->  Bitmap Heap Scan on accounts  (cost=19991.03..99447.38 rows=96637 width=19) (actual time=180.386..228.314 rows=719 loops=1)
        Recheck Cond: ((followers_count < 10) AND (friends_count > 1000))
        Heap Blocks: exact=718
        ->  BitmapAnd  (cost=19991.03..19991.03 rows=96637 width=0) (actual time=176.341..176.342 rows=0 loops=1)
              ->  Bitmap Index Scan on accounts_followers_count_bt  (cost=0.00..4273.59 rows=504927 width=0) (actual time=38.885..38.885 rows=511595 loops=1)
                    Index Cond: (followers_count < 10)
              ->  Bitmap Index Scan on accounts_friends_count_bt  (cost=0.00..15668.87 rows=1854018 width=0) (actual time=131.467..131.467 rows=1846112 loops=1)
                    Index Cond: (friends_count > 1000)
Planning Time: 2.113 ms
JIT:
  Functions: 4
  Options: Inlining false, Optimization false, Expressions true, Deforming true
  Timing: Generation 2.132 ms, Inlining 0.000 ms, Optimization 0.286 ms, Emission 2.709 ms, Total 5.128 ms
Execution Time: 232.939 ms
```

Podmínka: Nad *followers_count* a *friends_count* se automaticky vytvořily bitmap indexy nad btree. Oba indexy se použily při scanu vybraných stránek.

Seřazení: I po přidání btree se stále používá quicksort, protože je to výhodnější jak tvořit hashe.

15.

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

```
QUERY PLAN
Sort  (cost=93397.61..93639.20 rows=96637 width=19) (actual time=6.058..6.092 rows=719 loops=1)
  Sort Key: statuses_count
  Sort Method: quicksort  Memory: 69kB
  ->  Bitmap Heap Scan on accounts  (cost=5939.56..85395.92 rows=96637 width=19) (actual time=5.390..5.914 rows=719 loops=1)
        Recheck Cond: ((followers_count < 10) AND (friends_count > 1000))
        Heap Blocks: exact=718
        ->  Bitmap Index Scan on accounts_followers_friends_count_bt  (cost=0.00..5915.41 rows=96637 width=0) (actual time=5.329..5.329 rows=719 loops=1)
              Index Cond: ((followers_count < 10) AND (friends_count > 1000))
Planning Time: 0.643 ms
Execution Time: 6.168 ms
```

Tady se použil složený index na scanování stránky, o to je výsledek rychlejší.

16.

Upravte query tak, aby bol follower_count menší ako 1000 a friends_count väčší ako 1000. V čom je rozdiel a prečo?

```
QUERY PLAN
Sort  (cost=409422.93..412752.29 rows=1331744 width=19) (actual time=2144.876..2235.667 rows=740655 loops=1)
  Sort Key: statuses_count
  Sort Method: quicksort  Memory: 75388kB
  ->  Bitmap Heap Scan on accounts  (cost=81839.87..273952.03 rows=1331744 width=19) (actual time=165.095..1952.906 rows=740655 loops=1)
        Recheck Cond: ((followers_count < 1000) AND (friends_count > 1000))
        Heap Blocks: exact=169881
        ->  Bitmap Index Scan on accounts_followers_friends_count_bt  (cost=0.00..81506.93 rows=1331744 width=0) (actual time=133.699..133.700 rows=740655 loops=1)
              Index Cond: ((followers_count < 1000) AND (friends_count > 1000))
Planning Time: 0.532 ms
JIT:
  Functions: 4
  Options: Inlining false, Optimization false, Expressions true, Deforming true
  Timing: Generation 2.128 ms, Inlining 0.000 ms, Optimization 0.270 ms, Emission 2.691 ms, Total 5.089 ms
Execution Time: 2299.936 ms
```

Je tu mnohem víc heap_blocks. Musí se procházet více stránek, je to pomalejší.

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?

```
QUERY PLAN
Index Only Scan using accounts_screen_name_bt on accounts  (cost=0.43..2.65 rows=1 width=11) (actual time=11.302..11.306 rows=1 loops=1)
  Index Cond: (screen_name = 'realDonaldTrump'::text)
  Heap Fetches: 0
Planning Time: 0.156 ms
Execution Time: 11.339 ms
```

```
QUERY PLAN
Index Scan using accounts_screen_name_hash on accounts  (cost=0.00..2.22 rows=1 width=11) (actual time=0.709..0.712 rows=1 loops=1)
  Index Cond: ((screen_name)::text = 'realDonaldTrump'::text)
Planning Time: 0.331 ms
Execution Time: 0.742 ms
```

Chceme přesnou shodu. Hash index je rychlejší, jeslikož má časovou složitost O(1). Pokud necháme oba indexy, vybere se rychlejší hash.

```
QUERY PLAN
Index Only Scan using accounts_screen_name_bt on accounts  (cost=0.43..2.65 rows=1 width=11) (actual time=0.144..0.145 rows=0 loops=1)
  Index Cond: (screen_name = 'ldonaldt'::text)
  Heap Fetches: 0
Planning Time: 0.227 ms
Execution Time: 0.171 ms
```

```
QUERY PLAN
Index Scan using accounts_screen_name_hash on accounts  (cost=0.00..2.22 rows=1 width=11) (actual time=0.315..0.316 rows=0 loops=1)
  Index Cond: ((screen_name)::text = 'ldonaldt'::text)
Planning Time: 0.421 ms
Execution Time: 0.343 ms
```

U hash indexu je čas ještě nižší, jelikož nemusel nikam přistupovat, když zjistil, že *screen_name* neexistuje.

U btree je také čas nižší o přístup k datům.

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 zoraďte 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.

1. Vytvoření pomocných sloupců ts_vector

```
UPDATE tweets   37 m 57 s
SET content_vector = to_tsvector('english', content);

UPDATE accounts   waiting
SET name_description_vector = to_tsvector('english', name || ' ' || description);
```

2. Vytvoření indexu

```
CREATE INDEX tweets_contenet_gin ON tweets USING gin(content_vector);
CREATE INDEX accounts_name_descriptoin_gin ON accounts USING gin(name_description_vector);
```

3. Vytvoření query

```
SELECT t.content, a.description, a.name
FROM accounts a
         join tweets t on a.id = t.author_id
WHERE t.content_vector @@ to_tsquery('John & Oliver')
   or a.name_description_vector @@ to_tsquery('John & Oliver');
```

Více doladit query jsem nestla kvůli délce vytváření indexů a vektorů.