Fakulta informatiky a informačných technológií
STU Bratislava

Pokročilé databázové technológie

Protokol - Zadanie 2

Jakub Povinec
https://github.com/kuko6/bulk-tweets-import/tree/main/docs

1. 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?

```
SELECT * FROM authors WHERE username = 'mfa_russia';
```

Fig. 1 - SQL pre prvú úlohu

Plánovač vybral *Parallel Sequential Scan* pretože pre daný dopyt neexistuje v tabuľke žiaden vhodný index a keďže je povolené aj použitie paralelne spracovania, ktoré pokladal sa rýchlejšie.

```
1  Gather  (cost=1000.00..134997.51 rows=1 width=127) (actual time=125.092..128.377 rows=1 loops=1)
2    Workers Planned: 4
3    Workers Launched: 4
4    ->  Parallel Seq Scan on authors  (cost=0.00..133997.41 rows=1 width=127) (actual time=97.945..117.849 rows=0 loops=5)
5          Filter: ((username)::text = 'mfa_russia'::text)
6          Rows Removed by Filter: 1179035
7  Planning Time: 0.178 ms
8  Execution Time: 128.412 ms
```

Fig. 2 - EXPLAIN ANALYSE z prvej úlohy

	id	name	username	description	followers_count	following_count	tweet_count	listed_count
1	255471924	MFA Russia	mfa_russia	Ministry of Foreign Affairs of Russia (Official account) | Country's account @Russia...	556912	1443	69213	5300

Fig. 3 - Výsledok z prvej úlohy

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 (napíšte a popíšte všetky parametre)?

Na selecte z prvej úlohy pracovali *4* workery, ktorí slúžia na paralelizáciu vstupnej query tak, aby na nej mohlo pracovať viac procesov naraz. Maximálny počet workerov ovplyvňuje premenná `max_parallel_workers_per_gather`, ktorá bola v mojom prípade nastavená na *4*. Počet workerov sa dá zmeniť pomocou:

```
SET max_parallel_workers_per_gather TO <pozadovany_pocet_workerow>;
```

Fig. 4 - SQL pre zmenenie maximálneho počtu workerov

V mojom prípade sa už so zvyšovaním maximálneho počtu workerov nemenil konečný čas. Zmenu, ale bolo možné pozorovať pri použití **menšieho** počtu workerov, kedy napr. pri použití 2 workerov trvalo vykonanie danej query 158ms (o 30ms viac ako pri použití 4 workerov).

```
1  Gather  (cost=1000.00..147352.14 rows=1 width=127) (actual time=29.179..158.349 rows=1 loops=1)
2    Workers Planned: 2
3    Workers Launched: 2
4    ->  Parallel Seq Scan on authors  (cost=0.00..146352.04 rows=1 width=127) (actual time=108.170..149.761 rows=0 loops=3)
5          Filter: ((username)::text = 'mfa_russia'::text)
6          Rows Removed by Filter: 1965058
7  Planning Time: 0.162 ms
8  Execution Time: 158.382 ms
```

Fig. 5 - EXPLAIN ANALYSE pri použití 2 workerov na query z prvej úlohy

Maximálny počet workerov musí byť menší ako sú hodnoty premenných `max_worker_processes` a `max_parallel_workers` a taktiež musí byť menší ako počet CPU.

3. 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?

V tomto prípade netreba špecifikovať typ indexu, keďže pri vytváraní indexov v Postgres je práve btree predvolený typ indexu.

```
CREATE INDEX authors_username ON authors(username);
```

Fig. 6 - SQL pre vytvorenie btree indexu pre stĺpec username v tabuľke authors

Použitím indexu sa čas vykonania vyhľadávania výrazne zlepšil. Je to spôsobené tým, že v tomto prípade sa prehľadáva vyvážená stromová štruktúra, v ktorej sa môže veľa záznamov na základe podmienky hneď vyradiť a samotné vyhľadávanie je tak veľmi efektívne. Ako je vidieť aj na výstupe z `EXPLAIN ANALYSE` pre danú query (Fig. 7 a Fig. 8), tento dopyt nebol vykonaný paralelne a teda zvýšenie počtu workerov nemá vplyv na rýchlosť vykonania danej query.

```
1 Index Scan using authors_username on authors  (cost=0.43..2.65 rows=1 width=127) (actual time=0.045..0.048 rows=1 loops=1)
2   Index Cond: ((username)::text = 'mfa_russia'::text)
3 Planning Time: 0.204 ms
4 Execution Time: 0.086 ms
```

Fig. 7 - EXPLAIN ANALYSE pre query z prvej úlohy po vytvorení indexu s použitím 4 workerov

```
1 Index Scan using authors_username on authors  (cost=0.43..2.65 rows=1 width=127) (actual time=0.052..0.055 rows=1 loops=1)
2   Index Cond: ((username)::text = 'mfa_russia'::text)
3 Planning Time: 0.181 ms
4 Execution Time: 0.086 ms
```

Fig. 8 - EXPLAIN ANALYSE pre query z prvej úlohy po vytvorení indexu s použitím 2 workerov

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

```
SELECT * FROM authors WHERE followers_count >= 100 AND followers_count <= 200;
```

Fig. 9 - SQL pre prvú časť štvrtej úlohy

```
1 Seq Scan on authors  (cost=0.00..204075.64 rows=764526 width=127) (actual time=0.019..400.072 rows=760088 loops=1)
2   Filter: ((followers_count >= 100) AND (followers_count <= 200))
3   Rows Removed by Filter: 5135088
4 Planning Time: 0.226 ms
5 Execution Time: 414.572 ms
```

Fig. 10 - EXPLAIN ANALYSE pre prvú query zo štvrtej úlohy

```
SELECT * FROM authors WHERE followers_count >= 100 AND followers_count <= 120;
```

Fig. 11 - SQL pre druhú časť štvrtej úloh

```
Gather  (cost=1000.00..158892.81 rows=201379 width=127) (actual time=0.355..150.957 rows=199937 loops=1)
  Workers Planned: 4
  Workers Launched: 4
  ->  Parallel Seq Scan on authors  (cost=0.00..137754.91 rows=50345 width=127) (actual time=0.030..131.013 rows=39987 loops=5)
        Filter: ((followers_count >= 100) AND (followers_count <= 120))
        Rows Removed by Filter: 1139048
Planning Time: 0.212 ms
Execution Time: 157.049 ms
```

Fig. 12 - EXPLAIN ANALYSE pre druhú query zo štvrtej úlohy

Prvý dopyt sa vykonával dlhšie, čo je spôsobené tým, že sa v tomto prípade prehľadáva väčší interval a plánovač použil klasický *Sequential Scan* namiesto paralelneho. Rozhodnutie plánovača sa dá odvodniť tým, že keď dopyt vráti veľké množstvo riadkov, cena takzvanej Gather operácie, kde sa "zhromažďujú" výsledky ostatných procesov, je veľmi veľká a daný dopyt sa preto neoplatí vykonávať paralelne.

	id	name	username	description
1	1391403261748338695	SJ Wealth	SjWealth	Financial Planner with specialty in wealth creation and risk management....
2	1204043515505729537	กิ้งก่องแก้วกุ๊กกิ๊กก๊อกแก๊ก/มัมหมีน้องเจน	qptt14	ไอค่อนไม่น่าเก็บวายจีมาจากถังขยะเลย
3	369490089	Zehra Aydın	ZehraAyd	İstanbul Sultan Selim Mesleki ve Teknik Anadolu Lisesi...
4	1107461437918633984	Chava🦎	_chavafloresr	:p
5	1142639009962721280	Emilio Lara	cheche28061	ig:cheche2806
6	1350076379270967297	Ilhan Güneşer	IlhanGuneser	EMPTY
7	3344881810	Mohamed Fahs	fahs77	Welcome!!
8	1399398622974861317	Code	Code0047	RT and Like are for the record keeping
9				

Data | Message | Chart | 1.697 s | 760,088 rows | Export...

Fig. 13 - Výsledok pre prvú query zo štvrtej úlohy

	id	name	username	description
1	1420186765705502720	Тимофеич	BFJ4KmSblffIqLE	EMPTY
2	1167033716	Anichu	Anichu_u	19. Auroner de corazaõ. Sigo a rubius desde el 2013 y aquí estamoh. Viciada a la u
3	816767174	Æ Able	aeAble05	The limits of tyrants are prescribed by the endurance of those whom they oppress.
4	55795174	mrafieq	mrafieq	EMPTY
5	1493830171	hasan zorlu	hzorlu83	EMPTY
6	1106117393716142080	😊😊😊	MuskaanJain27	Free-spirited girl with an unpopular opinion....
7	1324025570880049153	Nielen	niele_n	Creating my own sunshine.
8	1441413261455331335	micah	MLM0695	Dwight Shrute stan Account. Ive been to space. top tier taste in music. artist....
9				

Data | Message | Chart | 458 ms | 199,937 rows | Export...

Fig. 14 - Výsledok pre druhú query zo štvrtej úlohy

5. 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?

```
CREATE INDEX authors_followers_count ON authors(followers_count);
```

Fig. 15 - SQL pre vytvorenie btree indexu pre stĺpec followers_count v tabuľke authors

```
Bitmap Heap Scan on authors  (cost=8602.23..135732.33 rows=765473 width=127) (actual time=69.033..160.600 rows=760088 loops=1)
  Recheck Cond: ((followers_count >= 100) AND (followers_count <= 200))
  Heap Blocks: exact=115116
  ->  Bitmap Index Scan on authors_followers_count  (cost=0.00..8410.86 rows=765473 width=0) (actual time=53.204..53.204 rows=760088 loops=1)
        Index Cond: ((followers_count >= 100) AND (followers_count <= 200))
Planning Time: 0.411 ms
Execution Time: 176.203 ms
```

Fig. 16 - EXPLAIN ANALYSE pre prvú query zo štvrtej úlohy s použitím indexu

```
1  Bitmap Heap Scan on authors  (cost=2266.82..113407.01 rows=201687 width=127) (actual time=41.484..97.799 rows=199937 loops=1)
2    Recheck Cond: ((followers_count >= 100) AND (followers_count <= 120))
3    Heap Blocks: exact=94533
4    ->  Bitmap Index Scan on authors_followers_count  (cost=0.00..2216.40 rows=201687 width=0) (actual time=26.258..26.258 rows=199937 loops=1)
5          Index Cond: ((followers_count >= 100) AND (followers_count <= 120))
6  Planning Time: 0.293 ms
7  Execution Time: 102.235 ms
```

Fig. 17 - EXPLAIN ANALYSE pre druhú query zo štvrtej úlohy s použitím indexu

Vyhľadávanie bolo po použití indexov v oboch prípadoch efektívnejšie. Bitmap index je efektívny v prípade, ak dopyt vracia veľa záznamov, pretože sa nemusia všetky hneď načítať do pamäte. Na vytvorenie samotnej bitmapy slúži *Bitmap Index Scan*, kedy sa skontrolujú jednotlivé stránky. Následne sa pomocou *Bitmap Heap Scan* začnú čítať relevatné stránky z pamäte, tu je taktiež potrebné skontrolovať ešte raz pôvodnú podmienku na čo slúži *Recheck Cond*, pretože jednotlivé stránky môžu obsahovať viac riadkov, z ktorých nie všetky musia spĺňať danú podmienku.

6. 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?

```sql
CREATE INDEX authors_name ON authors(name);
CREATE INDEX authors_followers_count ON authors(followers_count);
CREATE INDEX authors_description ON authors(description);
```

Fig. 18 - SQL pre vytvorenie btree indexov pre stĺpce name, followers_count a description v tabuľke authors

```sql
INSERT INTO authors VALUES
      (1, 'Jakub', 'jakub22', 'toto je description', 1000, 2, 3000, 400);
```

Fig. 19 - SQL pre vytvorenie záznamu do tabuľky authors

```sql
DROP INDEX authors_name;
DROP INDEX authors_followers_count;
DROP INDEX authors_description;
```

Fig. 20 - SQL pre dropnutie indexov pre stĺpce name, followers_count a description z tabuľky authors

Vkladanie do tabuľky s indexami je pomalšie pretože sa musia indexy aj aktualizovať, čo zahŕňa nájdenie správneho uzla, prípadné vyváženie stromu.

```
1  Insert on authors  (cost=0.00..0.01 rows=0 width=0) (actual time=2.791..2.793 rows=0 loops=1)
2    ->  Result  (cost=0.00..0.01 rows=1 width=1088) (actual time=0.004..0.004 rows=1 loops=1)
3  Planning Time: 0.086 ms
4  Execution Time: 2.832 ms
```

Fig. 21 - EXPLAIN ANALYSE pre vkladanie do authors s indexami

```
1  Insert on authors  (cost=0.00..0.01 rows=0 width=0) (actual time=0.385..0.386 rows=0 loops=1)
2    -> Result  (cost=0.00..0.01 rows=1 width=1088) (actual time=0.003..0.004 rows=1 loops=1)
3  Planning Time: 0.076 ms
4  Execution Time: 0.414 ms
```

Fig. 22 - EXPLAIN ANALYSE pre vkladanie do authors bez indexov

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

```
CREATE INDEX conversations_retweet_count ON conversations(retweet_count);
CREATE INDEX conversations_content ON conversations(content);
```

Fig. 23 - SQL pre vytvorenie btree indexov pre stĺpce retweet_count a content v tabuľke conversations

```
Query 1 OK: CREATE INDEX
```

Data | Message | Chart | 15.352 s | Query 1 OK: CREATE INDEX

Fig. 24 - vytvorenie indexu pre stĺpec retweet_count

```
Query 1 OK: CREATE INDEX
```

Data | Message | Chart | 31.894 s | Query 1 OK: CREATE INDEX

Fig. 25 - vytvorenie indexu pre stĺpec content

Vytváranie indexu pre stĺpec *content* je pomalšie pretože sa jedná o väčšie dáta, ktoré sa taktiež zložitejšie porovnávajú (čo je dôležité pri samotnom vytváraní indexu) narozdiel od čísel v prípade stĺpca *retweets_count*.

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

	index_name	tree_level	root_block_no	index_size	internal_pages	leaf_pages	avg_item_size
1	authors_name	3	21048	195403776	162	23690	105
2	conversations_content	5	142659	2517622784	12258	295068	173
3	authors_username	2	220	179576832	106	21814	22
4	conversations_retweet_count	2	209	225804288	136	27427	729
5	authors_description	4	46162	601890816	2021	71451	729
6	authors_followers_count	2	209	43294720	26	5258	729

Fig. 26 - Informácie o indexoch

```
CREATE EXTENSION pgstattuple;
CREATE EXTENSION pageinspect;

SELECT 'conversations_content' AS index_name, tree_level, root_block_no, index_size, internal_pages, leaf_pages, avg_item_size
    FROM pgstatindex('conversations_content'), bt_page_stats('conversations_content', 1)
UNION
SELECT 'conversations_retweet_count', tree_level, root_block_no, index_size, internal_pages, leaf_pages, avg_item_size
    FROM pgstatindex('conversations_retweet_count'), bt_page_stats('conversations_retweet_count', 1)
UNION
SELECT 'authors_username', tree_level, root_block_no, index_size, internal_pages, leaf_pages, avg_item_size
    FROM pgstatindex('authors_username'), bt_page_stats('authors_username', 1)
UNION
SELECT 'authors_name', tree_level, root_block_no, index_size, internal_pages, leaf_pages, avg_item_size
    FROM pgstatindex('authors_name'), bt_page_stats('authors_name', 1)
UNION
SELECT 'authors_description', tree_level, root_block_no, index_size, internal_pages, leaf_pages, avg_item_size
    FROM pgstatindex('authors_description'), bt_page_stats('authors_description', 1)
UNION
SELECT 'authors_followers_count', tree_level, root_block_no, index_size, internal_pages, leaf_pages, avg_item_size
    FROM pgstatindex('authors_followers_count'), bt_page_stats('authors_followers_count', 1);
```

Fig. 27 - SQL pre získanie informácií o indexoch

Z tabuľky môžeme vidieť, že indexy nad textovými stĺpcami obsahujú viac stránok, ich stromy sú hlbšie a celkovo sú ich indexy aj pamäťovo náročnejšie, čo je spôsobené tým, že samotné hodnoty sú väčšie a zložitejšie na porovnávanie.

9. 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?

```
SELECT * FROM conversations WHERE content LIKE '%Gates%';
```

Fig. 28 - SQL pre úlohu 9

```
1  Gather  (cost=1000.00..1120973.61 rows=57582 width=221) (actual time=4.906..11889.201 rows=4199 loops=1)
2    Workers Planned: 4
3    Workers Launched: 4
4    ->  Parallel Seq Scan on conversations  (cost=0.00..1114215.41 rows=14396 width=221) (actual time=3.359..11880.574 rows=840 loops=5)
5          Filter: (content ~~ '%Gates%'::text)
6          Rows Removed by Filter: 6468562
7  Planning Time: 2.344 ms
8  Execution Time: 11889.426 ms
```

Fig. 29 - EXPLAIN ANALYSE pre query z 9 úlohy bez použitia indexu

```
1  Gather  (cost=1000.00..1120973.61 rows=57582 width=221) (actual time=3.648..12051.813 rows=4199 loops=1)
2    Workers Planned: 4
3    Workers Launched: 4
4    ->  Parallel Seq Scan on conversations  (cost=0.00..1114215.41 rows=14396 width=221) (actual time=2.869..12042.241 rows=840 loops=5)
5          Filter: (content ~~ '%Gates%'::text)
6          Rows Removed by Filter: 6468562
7  Planning Time: 2.196 ms
8  Execution Time: 12052.051 ms
```

Fig. 30 - EXPLAIN ANALYSE pre query z 9 úlohy s indexom pre stĺpec content

Btree index nemá žiaden vplyv na vyhľadávanie pomocou `LIKE`, kde hladáme vzor na konci alebo v strede textu.

	id	author_id	content	possibly_sensitive	language	s
1	1496737708533686274	1262109623818977281	RT @BillinderGates: Run faster Joe 🏃‍♂️🏃‍♂️🏃‍♂️...	FALSE	en	Twitter
2	1496757774276988928	283512386	RT @BillinderGates: Run faster Joe 🏃‍♂️🏃‍♂️🏃‍♂️...	TRUE	en	Twitter
3	1496757972646596611	1264493853081550848	RT @BillinderGates: Run faster Joe 🏃‍♂️🏃‍♂️🏃‍♂️...	FALSE	en	Twitter
4	1496758256672260098	2992361364	RT @BillinderGates: Run faster Joe 🏃‍♂️🏃‍♂️🏃‍♂️...	FALSE	en	Twitter
5	1496746697241739265	612369289	RT @BillinderGates: Run faster Joe 🏃‍♂️🏃‍♂️🏃‍♂️...	FALSE	en	Twitter
6	1496753077487116289	1488114847568957442	RT @BillinderGates: Run faster Joe 🏃‍♂️🏃‍♂️🏃‍♂️...	FALSE	en	Twitter
7	1496739807879647234	1443457508824203267	RT @BillinderGates: Run faster Joe 🏃‍♂️🏃‍♂️🏃‍♂️...	FALSE	en	Twitter
8	1496740869956792325	1442367918671495174	RT @BillinderGates: Run faster Joe 🏃‍♂️🏃‍♂️🏃‍♂️...	FALSE	en	Twitter
9						

Data | Message | Chart | 11.329 s | 4,199 rows | Export...

Fig. 31 - Výsledok pre query z 9 úlohy

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

```
SELECT * FROM conversations
WHERE content LIKE 'There are no excuses%' AND possibly_sensitive;
```

Fig. 32 - SQL pre úlohu 10

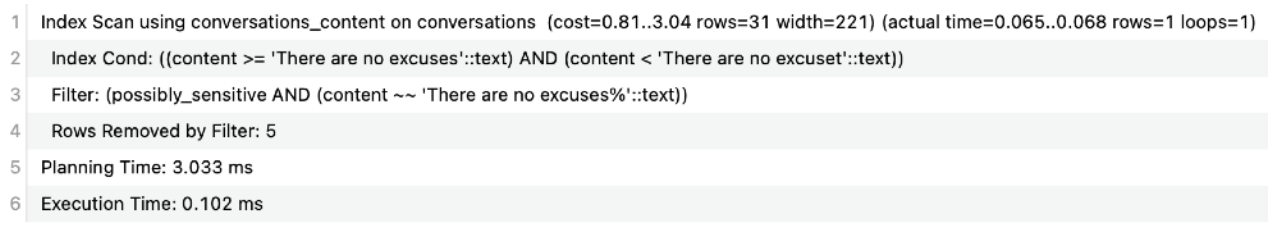


Fig. 33 - EXPLAIN ANALYSE pre query z 10 úlohy s indexom pre stĺpec content

V tomto prípade sa použil aj btree index pretože sa vzor nachádza na začiatku textu. Query by sa dalo ešte zefektívniť použitím iného druhu indexu ako GIN alebo GiST, ktoré sa viac hodia na prácu s textom.

Fig. 34 - Výsledok pre query z 10 úlohy

11. 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.

```
CREATE INDEX conversations_content_ending_url ON conversations (UPPER(
      RIGHT(conversations.content, LENGTH('https://t.co/pkFwLXZlEm'))));
```

Fig. 35 - SQL pre vytvorenie btree indexu v tabuľke conversations pre úlohu 11

Spôsob akým vytvoriť takýto btree index je na základe porovnania posledných *n* znakov, kde *n* je dĺžka daného reťazca. Na toto porovnanie slúži v Postgres funkcia `RIGHT()`, do ktorej idú ako argumenty string (v našom prípade stĺpec content) a počet znakov, ktoré chceme z daného stringu vrátiť. Kvôli tomu aby sa zhodovalo aj url napísané napr. len veľkými písmenami je potrebné ho pretransformovať buď na všetky písmená veľké alebo malé na čo slúži funkcia `UPPER()` resp. `LOWER()`. Rovnaký postup treba dodržať aj pri samotnom vyhľadávaní.

```
SELECT * FROM conversations conv
WHERE UPPER(RIGHT(conv.content, LENGTH('https://t.co/pkFwLXZlEm'))) =
UPPER('https://t.co/pkFwLXZlEm');
```

Fig. 36 - SQL pre úlohu 11

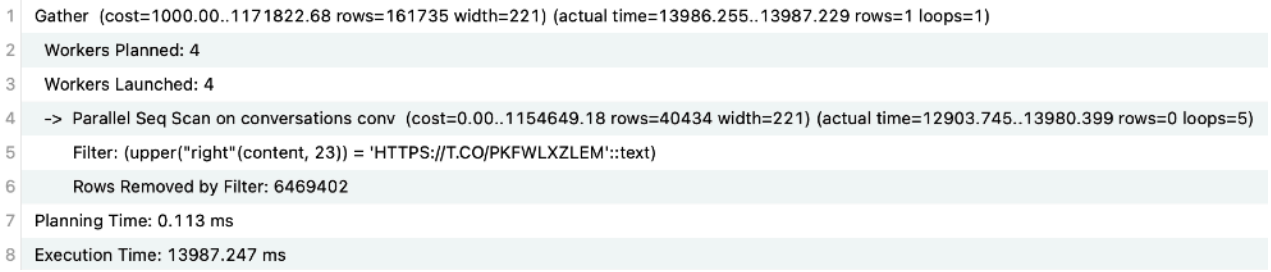


Fig. 37 - EXPLAIN ANALYSE pre query z 11 úlohy bez indexu

```
Bitmap Heap Scan on conversations conv  (cost=1633.51..163462.86 rows=161735 width=221) (actual time=0.075..0.076 rows=1 loops=1)
  Recheck Cond: (upper("right"(content, 23)) = 'HTTPS://T.CO/PKFWLXZLEM'::text)
  Heap Blocks: exact=1
  ->  Bitmap Index Scan on conversations_content_ending_url  (cost=0.00..1593.08 rows=161735 width=0) (actual time=0.060..0.060 rows=1 loops=1)
        Index Cond: (upper("right"(content, 23)) = 'HTTPS://T.CO/PKFWLXZLEM'::text)
Planning Time: 0.366 ms
Execution Time: 0.116 ms
```

Fig. 38 - EXPLAIN ANALYSE pre query z 11 úlohy s indexom

	id	author_id	content	possibly_sensitive	language	source
1	1502105081151311873	1501402131030364163	Vladimir Putin Russia vs. The Satanic New World Order depopulation agenda....	FALSE	en	Twitter for iPhone

Fig. 39 - Výsledok pre query z 11 úlohy

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

```
SELECT * FROM conversations
WHERE reply_count > 150 AND retweet_count >= 5000 ORDER BY quote_count;
```

Fig. 40 - SQL pre úlohu 12

```
Gather Merge  (cost=1135566.59..1136711.97 rows=9566 width=221) (actual time=10494.008..10497.026 rows=8364 loops=1)
  Workers Planned: 4
  Workers Launched: 4
  ->  Sort  (cost=1134566.53..1134572.51 rows=2392 width=221) (actual time=10486.560..10486.662 rows=1673 loops=5)
        Sort Key: quote_count
        Sort Method: quicksort  Memory: 693kB
        Worker 0:  Sort Method: quicksort  Memory: 746kB
        Worker 1:  Sort Method: quicksort  Memory: 535kB
        Worker 2:  Sort Method: quicksort  Memory: 1019kB
        Worker 3:  Sort Method: quicksort  Memory: 598kB
        ->  Parallel Seq Scan on conversations  (cost=0.00..1134432.29 rows=2392 width=221) (actual time=230.160..10485.930 rows=1673 loops=5)
              Filter: ((reply_count > 150) AND (retweet_count >= 5000))
              Rows Removed by Filter: 6467729
Planning Time: 0.307 ms
Execution Time: 10498.548 ms
```

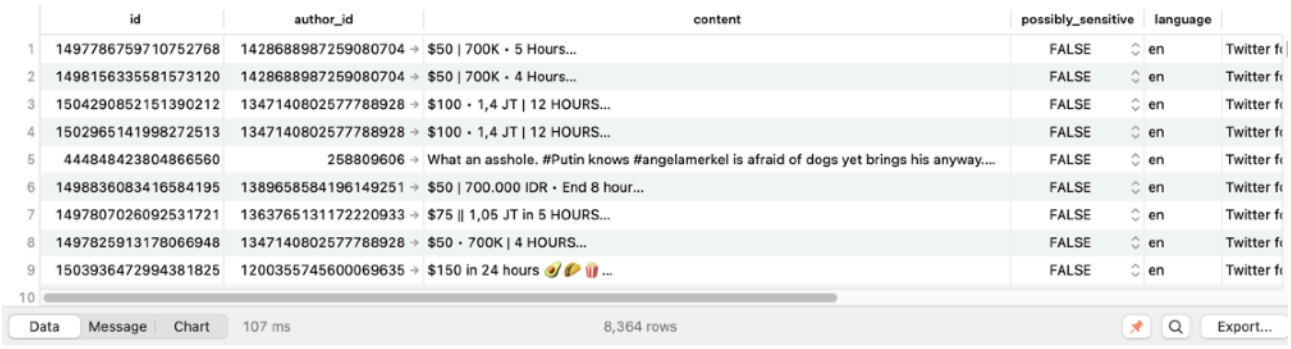
Fig. 41 - EXPLAIN ANALYSE pre query z 12 úlohy bez indexov

```
CREATE INDEX conversations_reply_count ON conversations (reply_count);
CREATE INDEX conversations_retweet_count ON conversations (retweet_count);
CREATE INDEX conversations_quote_count ON conversations (quote_count);
```

Fig. 42 - SQL pre vytvorenie btree indexov pre stĺpce reply_count, retweet_count a quote_count v tabuľke conversations

```
Sort  (cost=20762.50..20786.41 rows=9566 width=221) (actual time=78.160..78.458 rows=8364 loops=1)
  Sort Key: quote_count
  Sort Method: quicksort  Memory: 3684kB
  ->  Index Scan using conversations_reply_count on conversations  (cost=0.44..20130.01 rows=9566 width=221) (actual time=0.393..76.584 rows=8364 loops=1)
        Index Cond: (reply_count > 150)
        Filter: (retweet_count >= 5000)
        Rows Removed by Filter: 94248
Planning Time: 0.259 ms
Execution Time: 79.836 ms
```

Fig. 43 - EXPLAIN ANALYSE pre query z 12 úlohy s indexami



	id	author_id	content	possibly_sensitive	language	
1	1497786759710752768	1428688987259080704	$50 | 700K • 5 Hours...	FALSE	en	Twitter fo
2	1498156335581573120	1428688987259080704	$50 | 700K • 4 Hours...	FALSE	en	Twitter fo
3	1504290852151390212	1347140802577788928	$100 • 1,4 JT | 12 HOURS...	FALSE	en	Twitter fo
4	1502965141998272513	1347140802577788928	$100 • 1,4 JT | 12 HOURS...	FALSE	en	Twitter fo
5	444848423804866560	258809606	What an asshole. #Putin knows #angelamerkel is afraid of dogs yet brings his anyway....	FALSE	en	Twitter fo
6	1498836083416584195	1389658584196149251	$50 | 700.000 IDR • End 8 hour...	FALSE	en	Twitter fo
7	1497807026092531721	1363765131172220933	$75 || 1,05 JT in 5 HOURS...	FALSE	en	Twitter fo
8	1497825913178066948	1347140802577788928	$50 • 700K | 4 HOURS...	FALSE	en	Twitter fo
9	1503936472994381825	1200355745600069635	$150 in 24 hours 🥑🌮🍿...	FALSE	en	Twitter fo
10						

Data | Message | Chart | 107 ms | 8,364 rows | Export...

Fig. 44 - Výsledok pre query z 12 úlohy

Jediný index, ktorý sa použil pri danej query bol index pre stĺpec *reply_count*. Index sa použije na prvú časť podmienky, ktorá odstráni veľkú časť záznamov a pre ďalšiu časť podmienky plánovač uprednostní klasický filter.

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

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

Fig. 45 - SQL pre úlohu 13

```
1 Sort  (cost=7419.47..7443.39 rows=9566 width=221) (actual time=26.348..27.992 rows=8364 loops=1)
2   Sort Key: quote_count
3   Sort Method: quicksort  Memory: 3684kB
4   ->  Index Scan using conversations_multicol on conversations  (cost=0.56..6786.98 rows=9566 width=221) (actual time=0.111..20.414 rows=8364 loops=1)
5         Index Cond: ((reply_count > 150) AND (retweet_count >= 5000))
6 Planning Time: 0.383 ms
7 Execution Time: 28.673 ms
```

Fig. 46 - EXPLAIN ANALYSE pre query z 12 úlohy s použitím zloženého indexu

V tomto prípade sa použije index rovno na celý výraz bez nutnosti použiť filter, čo je nakoniec efektívnejšie ako použitie separátnych indexov.

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

V tejto úlohe som sa rozhodol použiť trigramy, ktoré v kombinácií s GiST a GIN indexami umožňujú použitie indexov aj pre `LIKE` v prípade, keď sa nenachádza len na začiatku. GiST index bol oproti GIN oveľa neefektívnejší. Dokonca bolo samotné vyhľadávanie pomalšie ako bez použitia indexu. Taktiež sa GiST index oveľa dlhšie vytváral a bol omnoho väčší (okolo 22GB oproti 5GB pre GIN), čo je spôsobené práve štruktúrou GiSTu, kedy pri trigramoch vznikne obrovský strom.

```sql
SELECT * FROM conversations
WHERE content LIKE '%Putin%New World Order%' AND possibly_sensitive;
```

Fig. 47 - SQL pre úlohu 14

```sql
CREATE EXTENSION pg_trgm;

CREATE INDEX conversations_gist_trgm_content
ON conversations USING gist(content gist_trgm_ops);

CREATE INDEX conversations_gin_trgm_content
ON conversations USING gin(content gin_trgm_ops);
```

Fig. 48 - SQL pre vytvorenie trigram GiST a GIN indexov pre stĺpec content

Zabudol som screenshotnuť správu o vytvorení indexu, tak tu je aspoň screenshot z histórie:

Fig. 49 - História dopytov

```
Query 1 OK: CREATE INDEX
```

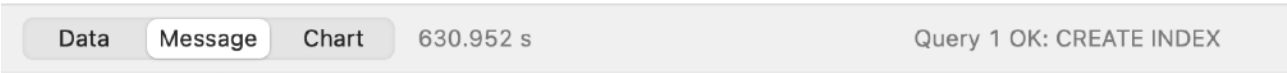


Fig. 50 - Vytvorenie GIN indexu

```
1 Index Scan using conversations_gist_trgm_content on conversations  (cost=0.55..3490.52 rows=31 width=221) (actual time=8386.540..186331.557 rows=5 loops=1)
2   Index Cond: (content ~~ '%Putin%New World Order%'::text)
3   Rows Removed by Index Recheck: 870
4   Filter: possibly_sensitive
5   Rows Removed by Filter: 221
6 Planning Time: 2.131 ms
7 Execution Time: 186331.886 ms
```

Fig. 51 - EXPLAIN ANALYSE pre GiST index

V prípade GiST sa použil klasický *Index Scan* spolu s filtrom pre vyradenie nevhodných záznamov kvôli stĺpcu *possibly_sensitive*. Query plán pre GIN index je zas rovnaký ako pri úlohe 11 kedy sa taktiež použil *Bitmap Index Scan*.

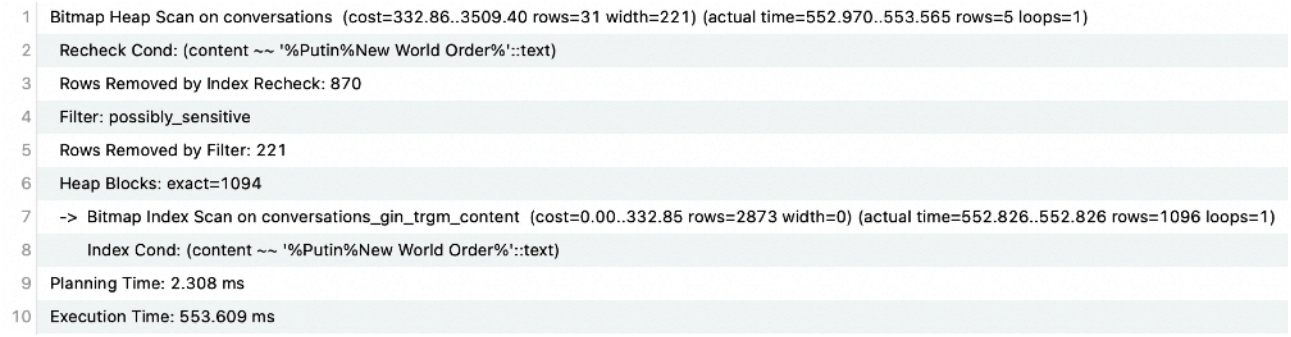


Fig. 52 - EXPLAIN ANALYSE pre GIN index

	id	author_id	content	possibly_sensitive	language	source
1	1498019842183544838	834033178067234816	RT @veteranstoday: Putin: New World Order Worships Satan...	TRUE	en	Twitter for A
2	1497653608028028931	1178005700188004355	#Putin and the New World Order from Damascus to Kiev...	TRUE	en	Twitter for A
3	1498341252927946758	1506482726	Would Putin take responsibility for this? 6 year old Ukrainian girl killed during Russian in...	TRUE	en	Twitter Web
4	1501745570045743104	3247724967	#Putin is gonna Destroy The New World Order! 🇷🇺 https://t.co/Vvha9YIwX4	TRUE	en	Twitter for iP
5	1498483618804649990	435312888	#Putin Has Banned #Rothschild And His New World Order Banking Cartel Family From...	TRUE	en	Twitter for A

Fig. 53 - Výsledok pre query zo 14 úlohy

15. 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.

V tomto prípade som sa rozhodol použiť rovnaký postup ako v predchádzajúcej úlohe (pretože je veľmi podobná), kedy som na vyhľadávanie použil `LIKE` operátor a trigram GIN index.

```
SELECT * FROM links WHERE url LIKE '%darujme.sk%';
```

Fig. 54 - SQL pre úlohu 15

```
CREATE INDEX links_gin_trgm_url ON links USING gin(url gin_trgm_ops);
```

Fig. 55 - SQL pre vytvorenie trigram GIN indexu pre stĺpec url

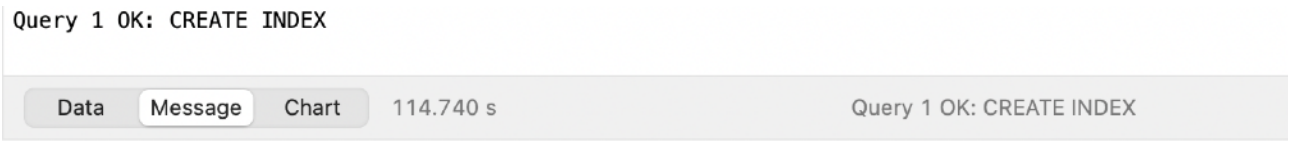


Fig. 56 - Vytvorenie GIN indexu pre 15 úlohu

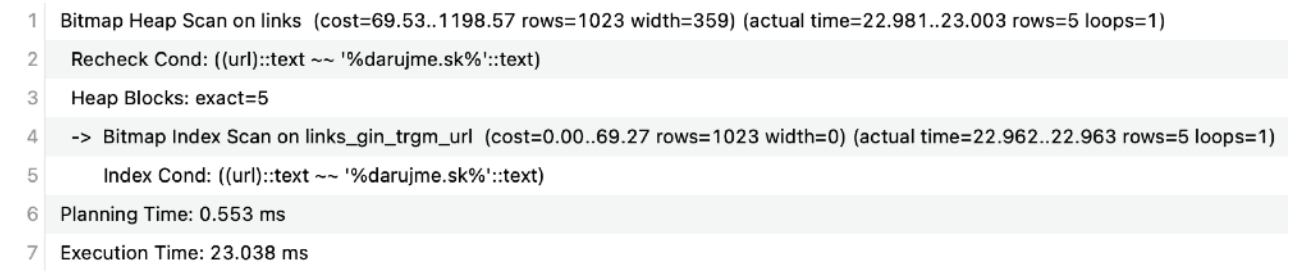


Fig. 57 - EXPLAIN ANALYSE pre GIN index z 15 úlohy

	id	conversation_id	url	title	
1	1590852	1497299831383076868	https://charita.darujme.sk/ukrajina/	Pomoc Ukrajine	To, čoho sme sa mesiace obávali, sa
2	4304874	1498345164833800192	https://clovekvohrozeni.darujme.sk/pomoc-ukrajina	NULL	NULL
3	9176936	1505620764636196870	https://redcross.darujme.sk/pomahame-ukrajine	Pomáhame Ukrajine	Prioritou Červeného kríža je zmierňo
4	10672875	1497548409334640644	https://redcross.darujme.sk/pomahame-ukrajine/	NULL	NULL
5	11036620	1501509972080877570	https://zvieraciombudsman.darujme.sk/animal-support-at-ukrainian-slovak-borders/	NULL	NULL

Fig. 58 - Výsledok pre query z 15 úlohy

Rovnako ako predtým aj v tomto prípade sa použil *Bitmap Index Scan* spolu s *Bitmap Heap Scan* a *Recheck Cond*.

16. 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.

Túto úlohu som nestihol dokončiť do funkčnej podoby. :(Moje pokusy sú ale v súbore `queries.sql`.