**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?**Obrázok, na ktorom je text

Automaticky generovaný popis

Použil som nasledujúcu query:

EXPLAIN ANALYZE SELECT username FROM authors

WHERE username = 'mfa\_russia';

Plánovač použil Parallel Seq Scan vybral ho preto, že som robil dopyt na neindexovaný stĺpec. Jedná sa o najmenej efektívny scan ktorí máme k dispozícii, lebo musíme pozrieť všetky záznamy tabuľky čo znamená veľa I/O operácii. Tie sa pri Seq Scan dopyte vykonávajú sekvenčne a nie náhodne preto je vykonanie rýchlejšie.

Výsledok:



**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)?**

Z obrázka vyššie môžeme vidieť, že na query pracovali dva workre. Worker je samostatný proces ktorý sa podieľa na vykonaní rovnakej query. Neparalelné časti query sú vykonané leader procesom. Počet workerov je limitovaný parametrami max\_parralel\_workers max\_worker\_processes a max\_paraller\_workers\_per\_gather. Zvýšil som počet max\_paralel\_workers\_per\_gather (nemôže byt väčší ako počet max\_worker\_processes) a následne zbehol ten istý select.

SET max\_parallel\_workers\_per\_gather = 4;

Obrázok, na ktorom je text

Automaticky generovaný popis

Planning time ako aj execution time sa skrátili execution time dokonca o polovicu. Strop nájdeme pri max\_parallel\_workers a max\_parallel\_workers\_per\_gather na čísle 1024 a pre max\_worker\_processes na 262143 ([8](#Eight)). Nastavenia týchto premenných by mali brať do úvahy dostupné HW zdroje max\_worker\_processses by nemalo byť viac ako je počet jadier na danom stroji.

Výsledok:



**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?**

Za pomoci nasledovného príkazu som vytvoril index v tabuľke authors nad stĺpcom username.

CREATE INDEX index\_username ON authors (

username

);

Vytvorenie indexu trvalo:



Obrázok, na ktorom je text

Automaticky generovaný popis

Z obrázku môžeme vidieť, že planing time zostal viac menej rovnaký avšak execution time sa neporovnateľne zrýchlil. Je to spôsobene tým, že teraz nemusíme prehľadávať celu tabuľku. Vieme sa rovno pozrieť kde sa nachádza hodnota ktorú hľadáme. V tejto požiadavke nepotrebujeme žiadneho workre pretože údaje ťaháme rovno z heapu.

Výsledok:



**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?**

Použil som query

EXPLAIN ANALYZE SELECT \* FROM authors

WHERE followers\_count >= 100 AND followers\_count <= 200

Obrázok, na ktorom je text

Automaticky generovaný popis

Použil som query

EXPLAIN ANALYZE SELECT \* FROM authors

WHERE followers\_count >= 100 AND followers\_count <= 120

Obrázok, na ktorom je text

Automaticky generovaný popis

Áno je tam rozdiel druha query je rýchlejšia je to logické pretože vyhľadávam v menšom číselnom rozsahu nad followers count. Takže prehľadávam menší počet údajov v druhej query sa dokonca použili aj workre.

Výsledok:

Obrázok, na ktorom je text

Automaticky generovaný popis

Obrázok, na ktorom je text, vnútri, niekoľko

Automaticky generovaný popis

**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?**

Vytvoril som nasledovný index

CREATE INDEX index\_followers\_count ON authors (

followers\_count

);

Čas trvania vytvorenia indexu bol:



Pre prvý prípad

Obrázok, na ktorom je text

Automaticky generovaný popis

Pre druhy prípad

Obrázok, na ktorom je text

Automaticky generovaný popis

Tým, že je stĺpec followers\_count zindexovaný tak je zoradený podľa hodnoty, to znamená, že riadky s podobnou hodnotou budú pri sebe čo urýchli načítanie.

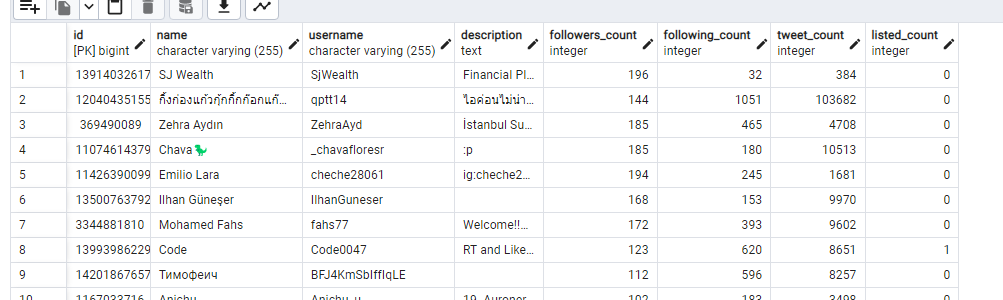
Bitmap Index Scan vytvorí bitmapu n-tic – pozícii konkrétnych riadkov (stránok ak to pamäť nedovolí), kde sa na základe indexu nachádzajú dáta, ktoré chcem z databázy získať

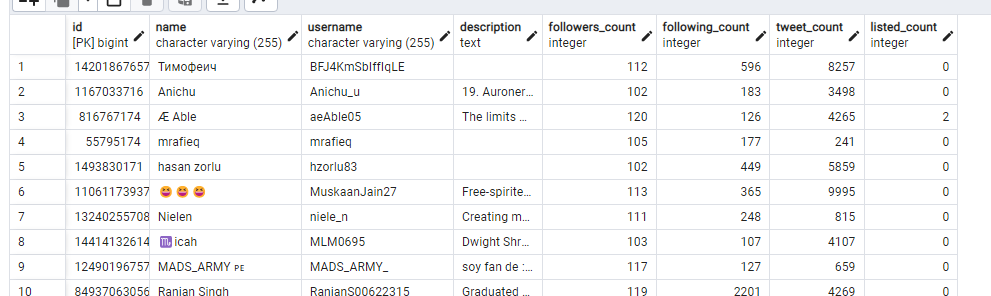
Bitmap Heap Scan následne použije túto vytvorenú bitmapu na zistenie ktoré stránky ma kontrolovať kvôli daným záznamom a ktoré môže preskočiť. Takto sa minimalizuje celkový počet I/O operácii a zvýši sa rýchlosť.

Re check condition: Pokiaľ sa ale bitmapa tvorená Bitmap Index Scanom stane priveľkou tak sa stane stratovou (pokiaľ to pamäť dovoľuje tak sa bitom označuje konkrétny záznam), ak je bitmapa stratová tak treba pre kontroval celu stránku ktorá je bitom reprezentovaná.

V explain analyze môžeme vidieť počet presných bitov v čísle HeapBlocks Exact a počet stratových nepresných stránkových bitov v čísle Heap Blocks lossy. ([1](#One), [2](#Two), [3](#Three))

Výsledok





**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?**

Vytvorenie indexov

CREATE INDEX index\_followers\_count ON authors (

followers\_count

);

CREATE INDEX index\_name ON authors (

"name"

);

CREATE INDEX index\_description ON authors (

description

);

Trvalo to :



Insert autora po pridaní indexov

Obrázok, na ktorom je text

Automaticky generovaný popis

Bez indexov

Insertol som pomocou nasledujúcej query

EXPLAIN ANALYZE INSERT INTO authors("name", username, description, followers\_count, following\_count, tweet\_count, listed\_count)

VALUES ('name', 'username', 'description', 1111, 1111, 1111, 1111);

Query trvala

Obrázok, na ktorom je text

Automaticky generovaný popis

Môžeme vidieť, že execution time pri pridaní bez indexov je rýchlejší zhruba o polovicu. Dôvodom je, že keď máme v databáze btree index a niečo pridáme tak tieto indexy sa musia prerátať. Index je dobrý len ak v query mame where. Operácia vloženia do tabuľky s indexami je zložitejšia pretože musí držať usporiadaný binárny strom a vložiť záznam na správne miesto. ([4](#Four))

Výsledok s indexami:

Obrázok, na ktorom je text

Automaticky generovaný popis

**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?**

Conversations retweet count

CREATE INDEX index\_retweet\_count ON conversations (

retweet\_count

);

Trvalo to

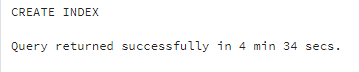


CREATE INDEX index\_content ON conversations (

"content"

);

Trvalo to:



Dĺžka indexu je ovplyvnená tým nad akým typom vytvárame index. retweet\_count je číslo pričom content je text. Text je omnoho náročnejší dátový typ ako integer. Dôvodom je, že pri tvorbe btree indexu sa porovnáva vždy hodnota záznamu s hodnotami ktoré sú už v strome a porovnať dve čísla trvá menej ako porovnať dva stringy. Zároveň na rozdiel od retweet\_count sa na content nedá použiť deduplikácia to znamená viac stránok a hlbší strom. Hlbší strom znamená dlhšiu cestu pre záznam pri triedení počas tvorby. Aj uloženie textu zaberá viac miesta na disku ako môžeme vidieť z nasledujúcich screenov.

Pomocou príkazu

select pg\_size\_pretty(pg\_total\_relation\_size('index\_content'));

Zistime, že veľkosť indexu pre content je nasledovná

Obrázok, na ktorom je stôl

Automaticky generovaný popis

Pomocou príkazu

select pg\_size\_pretty(pg\_total\_relation\_size('index\_retweet\_count'));

Zistime, že veľkosť indexu retweet count je

Obrázok, na ktorom je stôl

Automaticky generovaný popis

**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.**

Na zistenie týchto informácii som použil nasledujúce query:

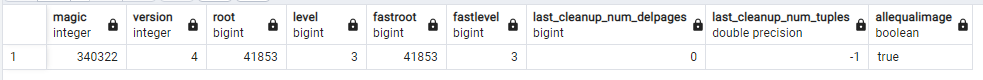
create extension pageinspect;

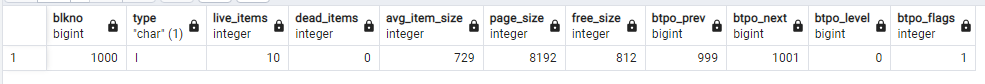
select \* from bt\_metap('index\_name');

select \* from bt\_page\_stats('index\_name', 1000);

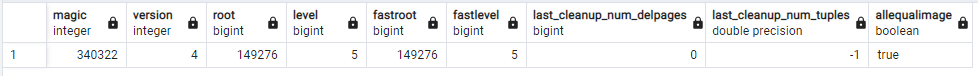
select \* from bt\_page\_items('index\_name', 1) limit 1000;

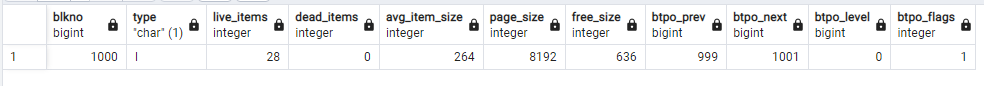
retweet\_count





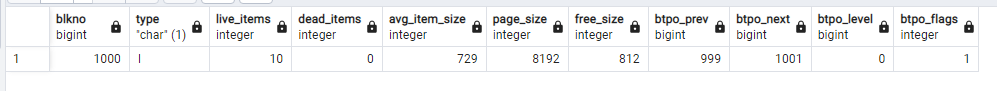
Content:



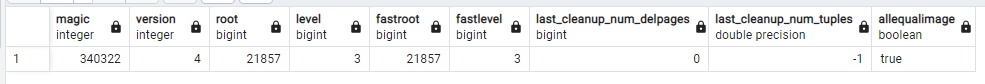


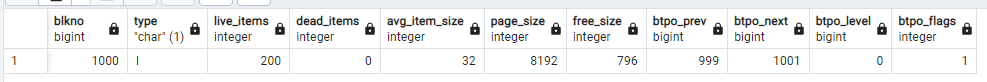
followers\_count:





Name





Počet root nodov je vždy ten istý keďže sa jedna o btree index a to je jedna. Content index ma na rovnakom počte záznamov väčšiu hĺbku stromu ako retweet\_count lebo pre varchar a text sa nemôže použiť deduplication tým pádom sa mení aj počet internal nodov. Avg\_item\_size ma prekvapil pretože by som očakával, že string bude zaberať viac ako integer ale opak je pravdou. Počet záznamov bol vo všetkých rovnaký. Každá tabuľka okrem name má dead\_items na nule to znamená, že v nich som nemazal data.

**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?**

Pomocou query som hľadal string gates:

SELECT \* from conversations

where content like '%GATES%'

Bez indexov tato operácia trvala

Obrázok, na ktorom je text

Automaticky generovaný popis

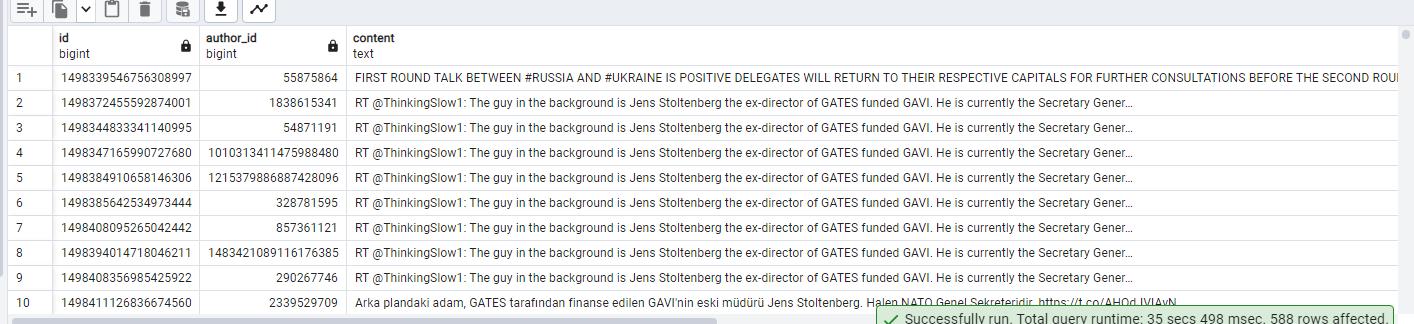
Po vytvorení indexov táto operácia trvala

Obrázok, na ktorom je text

Automaticky generovaný popis

Z explain analyze môžeme vidieť, že index nebol použitý. Index je dobrý na hľadanie konkrétnych veci napríklad meno = peter alebo daj mi všetky mená začínajúce sa na ‚pe‘ a končiace ľubovoľne alebo začínajúce sa ľubovoľne a končiace ‘pe’. Ale je zbytočný na query typu nájdi mi všetky mená ktoré obsahujú „pe“. Na to aby sme vedeli takto použiť index potrebujeme full text index. Môžeme použiť napríklad GIN. ([5](#Five)). Planing time v druhom bol dlhší pretože plánovač musel najprv vyhodnotiť či sa index oplatí použiť alebo nie a až potom vykonať query.

Výsledok:



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

Obrázok, na ktorom je text

Automaticky generovaný popis

Index sa nepoužil pretože mam zlý index pre operátor like. Použitím nasledujúceho príkazu vytvoríme index ktorý bude možne použiť.

CREATE INDEX tbl\_col\_text\_pattern\_ops\_idx ON conversations("content" text\_pattern\_ops);



Použil som query:

SELECT \* FROM conversations

Where content like 'There are no excuses%'

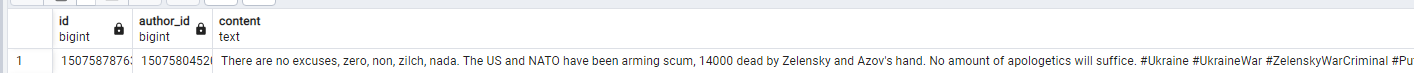
and possibly\_sensitive = TRUE;

Obrázok, na ktorom je text

Automaticky generovaný popis

Teraz bol index použitý a execution time je rýchly. ([6](#Six))

Výsledok

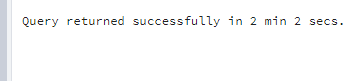


**11. Vytvorte nový btree index, tak aby ste pomocou neho vedeli vyhľadať tweet, ktorý končí reťazcom „https://t.co/pkFwLXZlEm“ kde nezáleží na tom ako to napíšete. Popíšte čo jednotlivé funkcie robia.**

Najprv potrebujem vytvoriť index pre opačné slová.

CREATE INDEX ON conversations (reverse(content) text\_pattern\_ops);

Čas trvania bol



Použil som query:

SELECT CONTENT FROM conversations

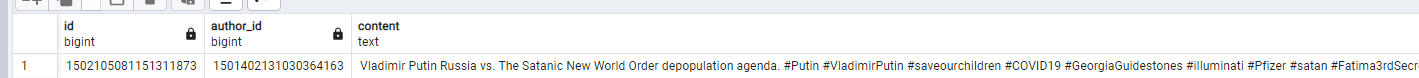
Where content like reverse('%https://t.co/pkFwLXZlE');

Obrázok, na ktorom je text

Automaticky generovaný popis

Teraz vidím, že sa použil index na vyhľadanie a našlo mi pravé jeden záznam.

Výsledok:



**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.**

Query:

EXPLAIN ANALYZE SELECT \* from conversations

where reply\_count > 150 and

retweet\_count >= 5000

ORDER BY quote\_count;

Bez indexov vyzerá plan takto:

Obrázok, na ktorom je text

Automaticky generovaný popis

Následne pomocou príkazu vytvorím index nad reply\_count

CREATE INDEX index\_reply\_count ON conversations (

reply\_count

);

Trvalo to

Obrázok, na ktorom je text

Automaticky generovaný popis

Obrázok, na ktorom je text

Automaticky generovaný popis

Samotne pridanie indexu nad reply\_count neurobilo nič pretože plánovač vyhodnotil, že sa ho neoplatí použiť.

Pomocou príkazu vytvorím index nad retweet\_count

CREATE INDEX index\_retweet\_count ON conversations (

retweet\_count

);

Trvalo to

Obrázok, na ktorom je text

Automaticky generovaný popis

Obrázok, na ktorom je text

Automaticky generovaný popis

Znova bol použitý sekvenčný scan takže index nebol použitý.

Pomocou príkazu vytvorím index nad quote\_count

CREATE INDEX index\_quote\_count ON conversations (

quote\_count

);

Trvalo to

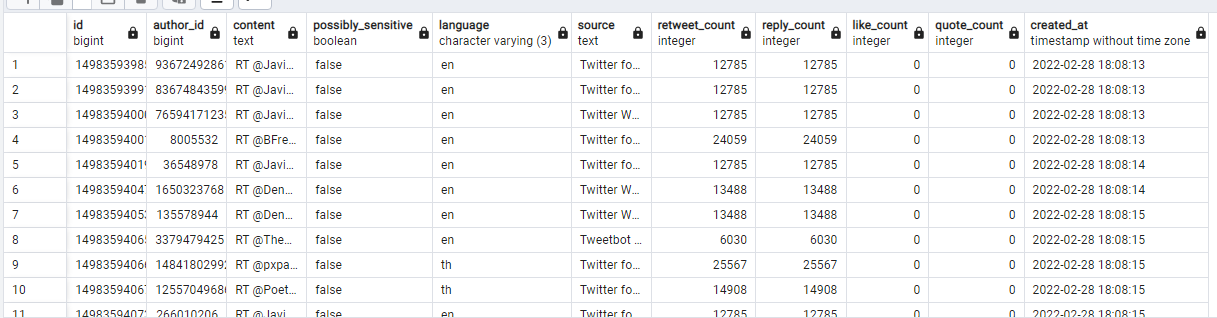


Obrázok, na ktorom je text

Automaticky generovaný popis

Každý krát sa použil sekvenčný scan namiesto použitia indexu takže všetky tieto tri mnou vytvorené indexy sú úplne zbytočne dokonca kontraproduktívne pretože zaberajú miesto na disku. Pri všetkých troch indexoch bolo nutne vyrátať dokopy tri bit mapy a až následne dokázal postgres použiť bitmap heap scan na vybratie potrebných záznamov. Dopyt avšak nad podmienkou vyhovoval len malo záznamom a tak musel prejsť všetky záznamy

Výsledok:



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

Pomocou nasledovného príkazu vytvorím zlozený index.

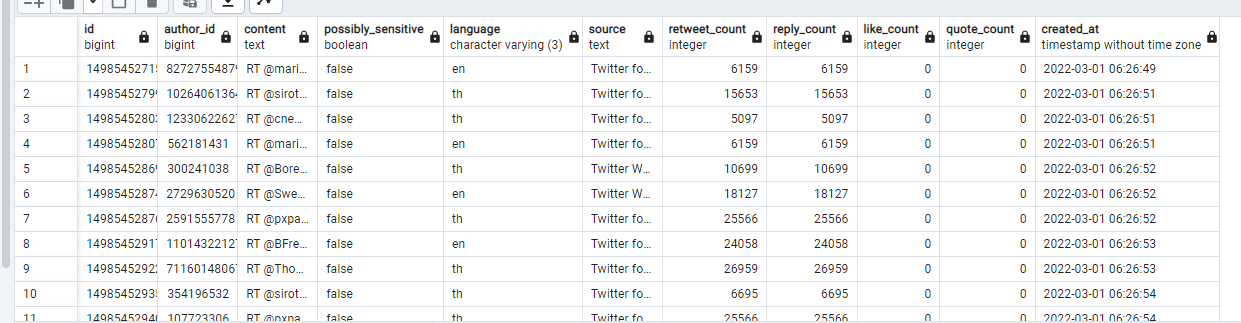
CREATE INDEX big\_index ON conversations(reply\_count, retweet\_count);

Obrázok, na ktorom je text

Automaticky generovaný popis

Výsledok je premena prekvapivý pretože by som čakal že vytvorením nasledovného zlozeného indexu sa case obrovsky zníži ale keďže planer použil paralel scan tak tento index je zbytočný. Takže medzi vytvorením samostatného indexu a vytvorením zložitého indexu nie je žiaden rozdiel

Výsledok:



**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?**

Pomocou nasledujúceho príkazu nainštalujem GIST a GIN

create extension pg\_trgm;

pomocou nasledovného príkazu vytvorím gist index nad contentom

CREATE INDEX tbl\_col\_gist\_trgm\_idx ON conversations USING gist (content gist\_trgm\_ops);

Pomocou nasledovného príkazu vytvorím gin index nad content

CREATE INDEX tbl\_col\_gin\_trgm\_idx ON conversations USING gin (content gin\_trgm\_ops);

Vytvorenie gin indexu trvalo



Vytvorenie gistu trvalo



Na nasledujúcom obrázku môžeme vidieť ako vyzerá query plan pre gin

Obrázok, na ktorom je text

Automaticky generovaný popis

Na nasledujúcom obrázku môžeme vidieť ako vyzerá query plan pre gist

Obrázok, na ktorom je text

Automaticky generovaný popis

Query

SELECT \* FROM conversations

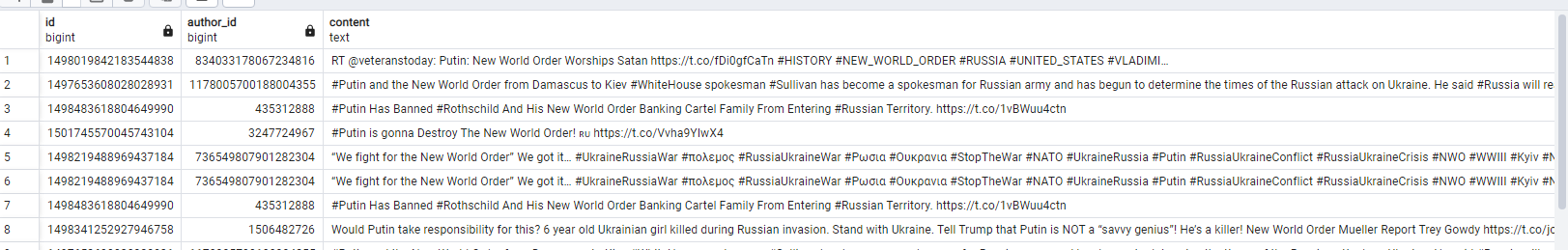
where content like '%Putin%'

and content like '%New World Order%'

and possibly\_sensitive = TRUE;

podľa obrázkov vyššie môžeme vidieť, že gin je neporovnateľne rýchlejší na vyhľadávanie či už to planning time alebo excution time. Gist zaberá aj viac miesta na disku ako gin. Avšak gin sa vytvára dlhšie ako gist.

Výsledok:



**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.**

Vytvorím indexy pomocou

CREATE INDEX tbl\_col\_gin\_trgm\_idx\_links ON links USING gin (url gin\_trgm\_ops);

CREATE INDEX tbl\_col\_gist\_trgm\_idx\_links ON links USING gist (url gist\_trgm\_ops);

Na dotaz použijem nasledujúcu query:

EXPLAIN ANALYZE SELECT \* FROM links

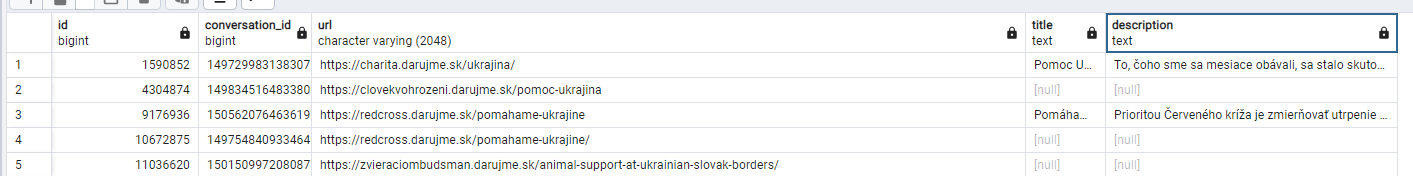
where url like '%darujme.sk%';

Ako môžeme vidieť bol použitý gin a to pre jeho vyššiu rýchlosť.

Obrázok, na ktorom je text

Automaticky generovaný popis

Výsledok:



**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 nachádzať̌ 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.**

Na začiatok som si potreboval vytvoriť indexy tie som vytvoril nasledovne:

Pomocou nasledovného príkazu som vytvoril gin index na username, description

CREATE INDEX index\_authors\_username\_description on authors USING GIN (to\_tsvector('english', username || ' ' || description));

pomocou nasledovného príkazu som vytvoril gin index na content

CREATE INDEX index\_conversations on conversations USING GIN (to\_tsvector('english', content));

pomocou nasledovného príkazu som vytvoril btree index nad author\_id

CREATE INDEX index\_author\_id ON conversations (

author\_id

);

Pomocou query

select a.username, a.description, t.content, t.retweet\_count from conversations t inner join authors a

on t.author\_id = a.id

and to\_tsvector('english', a.username || ' ' || a.description) @@ to\_tsquery('Володимир & Президент')

Union

select a.username, a.description, t.content, t.retweet\_count from authors a inner join conversations t

on a.id = t.author\_id

and to\_tsvector('english', t.content) @@ to\_tsquery('Володимир & Президент')

ORDER BY retweet\_count desc;

Select som rozdelil na dva samostatne selecty tým som sa vyhol OR podmienke ktorá nedokáže použiť indexy. Pri čiastkových selectoch som vždy prefiltroval jednu z tabuliek za použitia podmienky s tsvectorom a tsquery. Takto prefiltrovaná tabuľka bola v joine vždy naľavo pretože je menšia. Z týchto joinov som vždy vybral všetky stĺpce ktoré som potreboval. Výsledky daných čiastkových selectov som následne spojil pomocou príkazu union. Nakoniec som zoradil pomocou order by podľa stĺpca retweet count.

Obrázok, na ktorom je text

Automaticky generovaný popis

Výsledok:

Obrázok, na ktorom je text

Automaticky generovaný popis

**Literatúra:**

1. <https://pganalyze.com/docs/explain/scan-nodes/bitmap-index-scan#:~:text=Description%3A,easier%20to%20read%20in%20bulk>

1. <https://stackoverflow.com/questions/6592626/what-is-a-bitmap-heap-scan-in-a-query-plan>

1. <https://stackoverflow.com/questions/50959814/what-does-recheck-cond-in-explain-result-mean>

1. <https://use-the-index-luke.com/sql/dml/insert>

1. <https://stackoverflow.com/questions/30944245/why-is-an-index-not-used-on-a-like-query-with-wildcards>

1. <https://stackoverflow.com/questions/1566717/postgresql-like-query-performance-variations?fbclid=IwAR1LMn0OIILRlTviaxGJMga3SEF6EfAtoxRbJpNuk-Aao8I06eqW-Metdyg>

1. <https://stackoverflow.com/questions/28975517/difference-between-gist-and-gin-index/28976555#28976555>

1. <https://postgresqlco.nf/doc/en/param/max_worker_processes/>