Jakub Knežo Github repo = https://github.com/Kubislav/PDT-2-odovzdanie

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?

```
EXPLAIN (ANALYZE,BUFFERS) SELECT * FROM accounts
WHERE accounts.screen_name = 'realDonaldTrump'
```

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Gather (cost=1000.00..223597.62 rows=1 width=11) (actual time=0.560..1076.265 rows=1 loops=1)
2	[...] Workers Planned: 2
3	[...] Workers Launched: 2
4	[...] Buffers: shared hit=192 read=171944
5	[...] -> Parallel Seq Scan on accounts (cost=0.00..222597.52 rows=1 width=11) (actual time=666.404..1023.634 rows=0 loops=3)
6	[...] Filter: ((screen_name)::text = 'realDonaldTrump'::text)
7	[...] Rows Removed by Filter: 3229079
8	[...] Buffers: shared hit=192 read=171944
9	Planning Time: 0.112 ms
10	Execution Time: 1076.281 ms

Plánovač sa rozhodol pre Paralelný sekvenčný scan. Pre paralelný skan sa rozhodol kvôli tomu že ma viac workerov k dispozícii a sekvenčný pretože k tomu nemáme index preto prechádza každý záznam.

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?

```
SET max_parallel_workers_per_gather = 1000;
EXPLAIN (ANALYZE,BUFFERS) SELECT * FROM accounts
WHERE accounts.screen_name = 'realDonaldTrump'
```

Na danom selecte pracovali 2 workeri. Workeri slúžia na paralelné rozdelenie práce na danej query. Aj keď som nastavil **SET max_parallel_workers_per_gather = 1000;** tak si plánovač vytvoril 5 workerov. V tomto prípade to závisí od plánovača ktorý bude pridávať workerov len pokým ich pridávaním nezlepší čas spracovania. V tomto prípade sa vykonala táto query rýchlejšie vďaka 5 workerov ako v predchádzajúcej úlohe.

	QUERY PLAN text
1	Gather (cost=1000.00..197357.63 rows=1 width=11) (actual time=0.606..899.601 rows=1 loops=1)
2	[...] Workers Planned: 5
3	[...] Workers Launched: 5
4	[...] Buffers: shared hit=1824 read=170312
5	[...] -> Parallel Seq Scan on accounts (cost=0.00..196357.53 rows=1 width=11) (actual time=645.343..794.182 rows=0 loops=6)
6	[...] Filter: ((screen_name)::text = 'realDonaldTrump'::text)
7	[...] Rows Removed by Filter: 1614540
8	[...] Buffers: shared hit=1824 read=170312
9	Planning Time: 0.058 ms
10	Execution Time: 899.616 ms

Jakub Knežo Github repo = https://github.com/Kubislav/PDT-2-odovzdanie

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?

```
CREATE INDEX screen_name_idx ON accounts(screen_name);
EXPLAIN (ANALYZE,BUFFERS) SELECT * FROM accounts WHERE screen_name = 'realDonaldTrump';
```

Data Output | Explain | Messages | Notifications

	QUERY PLAN text
1	Index Scan using screen_name_idx on accounts (cost=0.43..8.45 rows=1 width=118) (actual time=0.141..0.142 rows=1 loops=1)
2	[...] Index Cond: ((screen_name)::text = 'realDonaldTrump'::text)
3	[...] Buffers: shared read=4
4	Planning:
5	[...] Buffers: shared hit=7 read=1
6	Planning Time: 1.376 ms
7	Execution Time: 0.157 ms

Výrazne sa zvýšil čas plánovania ale na druhej strane sa rapídne znížil čas vykonávania dokonca je tento čas ešte menší ako s workermi. Nepoužívali sa workeri keďže plánovač využil index scan.

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?

```
EXPLAIN (ANALYZE,BUFFERS) 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=1226829 width=118) (actual time=0.014..2676.296 rows=1269496 loops=1)
2	[...] Filter: ((followers_count >= 100) AND (followers_count <= 200))
3	[...] Rows Removed by Filter: 8417742
4	[...] Buffers: shared hit=3361 read=168775
5	Planning Time: 0.076 ms
6	Execution Time: 2717.501 ms

Správanie je iné ako v prvej aj tretej úlohe. Používa sa len sekvenčný scan. Rozdiel medzi 4kou a 3kou je v tom že v 3ke sa používa aj index scan ktorý nie je možné použiť keďže tu nemáme aplikovaný index. Plánovač sa rozhodol nepoužiť paralelné vyhľadávanie a nepoužíva workerov. Paralelný scan môže byť rýchlejší avšak pridáva časovú náročnosť kvôli rozdeľovaniu práce medzi jednotlivých workerov. V tomto prípade bolo z pohľadu plánovača výhodnejšie preto využiť len sekvenčný scan.

Jakub Knežo Github repo = https://github.com/Kubislav/PDT-2-odovzdanie

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?

```
CREATE INDEX followers_count_idx ON accounts(followers_count);
EXPLAIN (ANALYZE,BUFFERS) SELECT * FROM accounts WHERE followers_count >= 100 AND
followers_count<=200;
```

	QUERY PLAN text
1	Bitmap Heap Scan on accounts (cost=16871.43..310158.01 rows=1226829 width=118) (actual time=99.215..2572.706 rows=12694...
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	[...] Buffers: shared read=173033 written=2
6	[...] -> Bitmap Index Scan on followers_count_idx (cost=0.00..16564.73 rows=1226829 width=0) (actual time=92.189..92.190 rows=...
7	[...] Index Cond: ((followers_count >= 100) AND (followers_count <= 200))
8	[...] Buffers: shared read=1077
9	Planning:
10	[...] Buffers: shared hit=10 read=6 dirtied=2
11	Planning Time: 1.907 ms
12	Execution Time: 2611.272 ms

Bitmap Index Scan si vytvára bit mapu pričom každý bit reprezentuje stránku. Ak daná hodnota je na stránke tak bit má hodnotu 1. Taktiež môže bit ukazovať priamo na záznam ak je počet výsledných riadkov malý. V našom prípade máme Bitmap Heap Scan ktorý teraz pracuje len so stránkami ktoré majú bit hodnotu 1. Tento Bitmap Heap scan vráti stránky kde sa nachádzajú hodnoty ktoré vyhovujú podmienky where. Dôvod prečo sa vykonáva recheck condition je ten že sa ako finálny krok prejdú stránky ktoré majú bit 1 a výberu sa len tie hodnoty ktoré vyhovujú podmienke where.

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

```
EXPLAIN (ANALYZE,BUFFERS) 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=4327017 width=118) (actual time=0.489..2543.093 rows=4382646 loops=1)
2	[...] Filter: ((followers_count >= 100) AND (followers_count <= 1000))
3	[...] Rows Removed by Filter: 5304592
4	[...] Buffers: shared hit=16310 read=155826
5	Planning Time: 0.086 ms
6	Execution Time: 2669.934 ms

Rozdiel je v tom že plánovač sa rozhodol využiť sekvenčný scan namiesto bitmap heap scan. Dôvodom je najskôr to že plánovač usúdil že časová cena vyhľadávania cez sekvenčný scan je menšia ako cez bitmap heap scan.

Jakub Knežo Github repo = https://github.com/Kubislav/PDT-2-odovzdanie

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?

S indexmi

```
CREATE INDEX name_idx ON accounts(name);
CREATE INDEX friends_count_idx ON accounts(friends_count);
CREATE INDEX description_idx ON accounts(name);
EXPLAIN (ANALYZE,BUFFERS) INSERT INTO accounts(name,screen_name,description,
followers_count, friends_count, statuses_count)
VALUES ('Michael Jordan', 'Russian President', 'https://www.youtube.com/watch?v=wpyNaifejf0', 18,
489, 4900);
```

	QUERY PLAN text
1	Insert on accounts (cost=0.00..0.01 rows=1 width=888) (actual time=11.483..11.483 rows=0 loops=1)
2	[...] Buffers: shared hit=6 read=21 dirtied=6
3	[...] -> Result (cost=0.00..0.01 rows=1 width=888) (actual time=6.916..6.916 rows=1 loops=1)
4	[...] Buffers: shared hit=6 read=7 dirtied=1
5	Planning:
6	[...] Buffers: shared hit=1 read=2
7	Planning Time: 0.459 ms
8	Execution Time: 11.535 ms

```
DROP INDEX name_idx;
DROP INDEX friends_count_idx;
DROP INDEX description_idx;
EXPLAIN ANALYSE INSERT INTO accounts(name,screen_name,description, followers_count,
friends_count, statuses_count)
VALUES ('generic name', 'generic screen name', 'generic description', 889, 58, 785);
```

	QUERY PLAN text
1	Insert on accounts (cost=0.00..0.01 rows=1 width=888) (actual time=1.416..1.417 rows=0 loops=1)
2	[...] -> Result (cost=0.00..0.01 rows=1 width=888) (actual time=0.004..0.005 rows=1 loops=1)
3	Planning Time: 0.021 ms
4	Execution Time: 1.440 ms

Je očividné že pri vkladaní bez indexu sa táto query vykonala rychlejšie hlavne kvôli tomu že netrebalo vytvárať ďalšie 3 indexy.

Jakub Knežo Github repo = https://github.com/Kubislav/PDT-2-odovzdanie

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?

```
CREATE INDEX retweet_count_idx ON tweets(retweet_count);
```

```
CREATE INDEX

Query returned successfully in 2 min.
```

```
CREATE INDEX content_idx ON tweets(content);
```

```
CREATE INDEX

Query returned successfully in 7 min 31 secs.
```

Content je typu text, má väčšiu dĺžku kľúča a vytvára zložitejšie indexové štruktúry oproti retweet_count typu int, ktorý si vystačí s kratším kľúčom.

9. Porovnajte indexy pre retweet_count, content, followers_count, screen_name,... v čom sa líšia a prečo (stačí stručne)?
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 itemoffset, itemlen, data from bt_page_items('idx_content',1) limit 1000;

b. select * from bt_metap('idx_content');

```
SELECT * FROM bt_metap('retweet_count_idx');
```

Data Output | Explain | Messages | Notifications

	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('content_idx');
```

Data Output | Explain | Messages | Notifications

	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	87624	5	87624	5	0	-1	true

```
SELECT * FROM bt_metap('screen_name_idx');
```

	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	217	2	217	2	0	-1	true

Jakub Knežo Github repo = https://github.com/Kubislav/PDT-2-odovzdanie

SELECT * FROM bt_metap('retweet_count_idx');

	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

Môžeme vidieť že content ma veľmi odlišné hodnoty root a fastroot ako ostatné indexy najskôr preto že slúži na indexovanie textu.

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

SELECT type, live_items, dead_items, avg_item_size, page_size, free_size FROM bt_page_stats('retweet_count_idx',1000);

	type "char" (1)	live_items integer	dead_items integer	avg_item_size integer	page_size integer	free_size integer
1	l	10	0	729	8192	812

SELECT type, live_items, dead_items, avg_item_size, page_size, free_size FROM bt_page_stats('content_idx',1000);

Data Output Explain Messages Notifications

	type "char" (1)	live_items integer	dead_items integer	avg_item_size integer	page_size integer	free_size integer
1	l	27	0	271	8192	704

SELECT type, live_items, dead_items, avg_item_size, page_size, free_size FROM bt_page_stats('screen_name_idx',1000);

	type "char" (1)	live_items integer	dead_items integer	avg_item_size integer	page_size integer	free_size integer
1	l	279	0	22	8192	808

SELECT type, live_items, dead_items, avg_item_size, page_size, free_size FROM bt_page_stats('followers_count_idx',1000);

	type "char" (1)	live_items integer	dead_items integer	avg_item_size integer	page_size integer	free_size integer
1	l	10	0	729	8192	812

Vracia nám sumárne informácie o jednotlivých stránkach B-tree indexov.

Jakub Knežo Github repo = https://github.com/Kubislav/PDT-2-odovzdanie

d. select itemoffset, itemlen, data from bt_page_items('idx_content',1) limit 1000;

SELECT itemoffset, itemlen, data FROM bt_page_items('retweet_count_idx',1) LIMIT 1000;

Data Output | Explain | Messages | Notifications

	itemoffset smallint	itemlen smallint	data text
1	1	24	00 00 00 00 00 00 00 00
2	2	808	00 00 00 00 00 00 00 00
3	3	808	00 00 00 00 00 00 00 00
4	4	808	00 00 00 00 00 00 00 00
5	5	808	00 00 00 00 00 00 00 00
6	6	808	00 00 00 00 00 00 00 00
7	7	808	00 00 00 00 00 00 00 00
8	8	808	00 00 00 00 00 00 00 00
9	9	808	00 00 00 00 00 00 00 00
10	10	808	00 00 00 00 00 00 00 00

SELECT itemoffset, itemlen, data FROM bt_page_items('content_idx',1) LIMIT 1000;

	itemoffset smallint	itemlen smallint	data text
1	1	152	38 02 00 00 27 27 23 57 ...
2	2	232	7c 03 00 00 7f 54 65 6f 7...
3	3	64	69 27 27 27 27 68 74 74 ...
4	4	272	10 04 00 00 27 27 27 54 ...
5	5	360	6a 05 00 00 03 02 00 00 ...
6	6	304	8c 04 00 00 27 27 20 2d ...
7	7	152	2c 02 00 00 27 27 20 2e ...
8	8	200	e8 02 00 00 27 27 20 41 ...
9	9	232	78 03 00 00 27 27 20 43 ...
10	10	176	94 02 00 00 27 27 20 43 ...
11	11	56	59 27 27 20 43 61 73 73 ...
12	12	216	38 03 00 00 27 27 20 43 ...
13	13	232	7c 03 00 00 27 27 20 43 ...
14	14	96	ab 27 27 20 43 6f 72 6f 6...
15	15	120	df 27 27 20 64 75 65 20 ...
16	16	352	48 05 00 00 27 27 20 44 ...
17	17	96	a5 27 27 20 65 73 74 c3 ...
18	18	256	d8 03 00 00 27 27 20 68 ...
19	19	64	71 27 27 20 69 6b 20 6d ...
20	20	216	24 03 00 00 27 27 20 4a ...
21	21	104	c1 27 27 20 4c 27 c3 a9 ...
22	22	288	5c 04 00 00 27 27 20 4d ...
23	23	296	70 04 00 00 27 27 20 4e ...
24	24	296	74 04 00 00 27 27 20 4e ...
25	25	232	6c 03 00 00 27 27 20 50 ...
26	26	296	64 04 00 00 27 27 20 50 ...
27	27	40	37 27 27 20 73 65 20 76 ...
28	28	272	20 04 00 00 27 27 20 53 ...
29	29	320	d4 04 00 00 27 27 20 54 ...
30	30	224	60 03 00 00 27 27 20 55 ...
31	31	208	1c 03 00 00 27 27 20 d8 ...
32	32	360	78 05 00 00 27 27 20 e0 ...
33	33	256	c8 03 00 00 27 27 e2 80 ...
34	34	120	d3 27 27 23 43 6f 72 6f 6...
35	35	144	20 02 00 00 27 27 23 43 ...

Jakub Knežo Github repo = https://github.com/Kubislav/PDT-2-odovzdanie

SELECT itemoffset, itemlen, data FROM bt_page_items('screen_name_idx',1) LIMIT 1000;

1	1	24	17 5f 5f 5f 5f 5f 5f 43 6f 6c 74 00 00 00 00 00
2	2	16	05 5f 00 00 00 00 00 00
3	3	16	07 5f 5f 00 00 00 00 00
4	4	24	21 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 63 66
5	5	24	21 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 7a 65
6	6	24	21 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 30 32 38
7	7	24	21 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 41 42 31
8	8	24	21 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 41 4c 4b
9	9	24	21 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 63 72 61
10	10	24	21 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 47 69 6f
11	11	24	21 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 69 72 6f
12	12	24	21 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 6a 6b 67
13	13	24	21 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 6e 77 77
14	14	24	21 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 71 33 33
15	15	24	21 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 5f 75 6f 75

263 záznamov

SELECT itemoffset, itemlen, data FROM bt_page_items('followers_count_idx',1) LIMIT 1000;

	itemoffset smallint	itemlen smallint	data text
1	1	24	00 00 00 00 00 00 00 00
2	2	808	00 00 00 00 00 00 00 00
3	3	808	00 00 00 00 00 00 00 00
4	4	808	00 00 00 00 00 00 00 00
5	5	808	00 00 00 00 00 00 00 00
6	6	808	00 00 00 00 00 00 00 00
7	7	808	00 00 00 00 00 00 00 00
8	8	808	00 00 00 00 00 00 00 00
9	9	808	00 00 00 00 00 00 00 00
10	10	808	00 00 00 00 00 00 00 00

Nerozumiem prečo tam sú tie 808 je to niečo čo si tam hádže psql inak vidieť veľkosti dát a ich obsah.

Jakub Knežo Github repo = https://github.com/Kubislav/PDT-2-odovzdanie

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

```
EXPLAIN (ANALYZE,BUFFERS) SELECT *
FROM tweets WHERE tweets.content LIKE '%Gates%';
```

	QUERY PLAN text
1	Gather (cost=1000.00..1115712.89 rows=306098 width=271) (actual time=8.990..400217.019 rows=111886 loops=1)
2	[...] Workers Planned: 7
3	[...] Workers Launched: 7
4	[...] Buffers: shared read=1027003
5	[...] -> Parallel Seq Scan on tweets (cost=0.00..1084103.09 rows=43728 width=271) (actual time=24.439..400061.923 rows=13986 loops=8)
6	[...] Filter: (content ~~ '%Gates%'::text)
7	[...] Rows Removed by Filter: 3983021
8	[...] Buffers: shared read=1027003
9	Planning:
10	[...] Buffers: shared hit=75 read=19 dirtied=1
11	Planning Time: 31.056 ms
12	Execution Time: 400235.496 ms

S indexom

```
CREATE INDEX tweet_content_idx ON tweets(content);
EXPLAIN ANALYSE SELECT * FROM tweets WHERE tweets.content LIKE '%Gates%';
```

	QUERY PLAN text
1	Gather (cost=1000.00..1115712.89 rows=306098 width=271) (actual time=7.415..71508.047 rows=111886 loops=1)
2	[...] Workers Planned: 7
3	[...] Workers Launched: 7
4	[...] -> Parallel Seq Scan on tweets (cost=0.00..1084103.09 rows=43728 width=271) (actual time=7.157..71353.088 rows=13986 ...
5	[...] Filter: (content ~~ '%Gates%'::text)
6	[...] Rows Removed by Filter: 3983021
7	Planning Time: 4.371 ms
8	Execution Time: 71517.321 ms

Časovo vznikol rozdiel keďže index bolo potrebné vytvoriť avšak aj mimo neho bol použitý v oboch prípadoch Paralelný sekvenčný scan pretože úplne základný obyčajný index nepracuje s LIKE podmienkou ak vyhľadáva slova ktoré končia alebo začínajú na %.

Jakub Knežo Github repo = https://github.com/Kubislav/PDT-2-odovzdanie

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

```
EXPLAIN ANALYSE SELECT * FROM tweets
WHERE tweets.content LIKE 'The Cabel and Deep State%';
```

Rovnako ako v úlohe 10 základný index nepracuje s LIKE podmienkou pri slovách ktoré končia alebo začinaju na %.

	QUERY PLAN text
1	Gather (cost=1000.00..1085406.09 rows=3030 width=271) (actual time=4637.460..4643.445 rows=1 loops=1)
2	[...] Workers Planned: 7
3	[...] Workers Launched: 7
4	[...] -> Parallel Seq Scan on tweets (cost=0.00..1084103.09 rows=433 width=271) (actual time=4241.825..4507.508 rows=0 loop...
5	[...] Filter: (content ~~ 'The Cabel and Deep State%'::text)
6	[...] Rows Removed by Filter: 3997006
7	Planning Time: 5.009 ms
8	Execution Time: 4643.477 ms

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?

```
CREATE INDEX tweet_content_like_idx ON tweets(content TEXT_PATTERN_OPS);
EXPLAIN (ANALYZE,BUFFERS) SELECT * FROM tweets WHERE tweets.content LIKE 'The Cabel and
Deep State”%'
```

	QUERY PLAN text
1	Index Scan using tweet_content_like_idx on tweets (cost=0.81..8.83 rows=3030 width=271) (actual time=0.083..0.083 rows=0 lo...
2	[...] Index Cond: ((content ~>=~ 'The Cabel and Deep State”'::text) AND (content ~<~ 'The Cabel and Deep State„'::text))
3	[...] Filter: (content ~~ 'The Cabel and Deep State”%'::text)
4	[...] Buffers: shared read=6
5	Planning:
6	[...] Buffers: shared hit=33 read=11 dirtied=1
7	Planning Time: 6.967 ms
8	Execution Time: 0.102 ms

```
EXPLAIN ANALYSE SELECT * FROM tweets WHERE tweets.content LIKE '%Gates%'
```

Jakub Knežo Github repo = https://github.com/Kubislav/PDT-2-odovzdanie

	QUERY PLAN text
1	Gather (cost=1000.00..1115712.89 rows=306098 width=271) (actual time=1.290..6130.653 rows=111886 loops=1)
2	[...] Workers Planned: 7
3	[...] Workers Launched: 7
4	[...] Buffers: shared hit=1152 read=1025851
5	[...] -> Parallel Seq Scan on tweets (cost=0.00..1084103.09 rows=43728 width=271) (actual time=2.957..5988.951 rows=13986 l...
6	[...] Filter: (content ~~ '%Gates%'::text)
7	[...] Rows Removed by Filter: 3983021
8	[...] Buffers: shared hit=1152 read=1025851
9	Planning Time: 0.190 ms
10	Execution Time: 6140.980 ms

Na to aby sa použil Index pri LIKE podmienke bolo potrebné po upraviť index pomocou TEXT_PATTERN_OPS. V prípade GATES sa index nepoužil keďže ide o infix a vyhľadávanie Infixu cez index je najskôr pre plánovač stále príliš časovo náročné na použitie preto sa rozhodol pre paralel seq scan.

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.

```
DROP INDEX tweet_content_like_idx;
CREATE INDEX tweets_content_like_case_insensitive_idx ON tweets(reverse(lower(content)) TEXT_PATTERN_OPS);
EXPLAIN ANALYSE SELECT * FROM tweets WHERE reverse(lower(content)) LIKE reverse(lower('%idiot #QAnon“'));
```

	QUERY PLAN text
1	Gather (cost=10667.58..538187.73 rows=159880 width=271) (actual time=118.145..377.855 rows=0 loops=1)
2	[...] Workers Planned: 5
3	[...] Workers Launched: 5
4	[...] -> Parallel Bitmap Heap Scan on tweets (cost=9667.58..521199.73 rows=31976 width=271) (actual time=0.061..0.061 rows=...
5	[...] Filter: (reverse(lower(content)) ~~ '"nonaq# toidi%'::text)
6	[...] -> Bitmap Index Scan on tweets_content_like_case_insensitive_idx (cost=0.00..9627.61 rows=159880 width=0) (actual time=...
7	[...] Index Cond: ((reverse(lower(content)) ~>=~ '"nonaq# toidi'::text) AND (reverse(lower(content)) ~<~ '"nonaq# toidj'::text))
8	Planning Time: 5.694 ms
9	Execution Time: 377.888 ms

Kedže nezáleží na tom ako to napíšeme využijeme lower a reverse aby sme dostali čo najviac hits. V tomto prípade Bitmap Index Scan prechádza indexom a zisťuje ktoré dáta je potrebné vytiahnuť. Vytvorí bitmapu ktorú následne pošle Bitmap Heap Scan ktorý túto mapu dekóduje a vytiahne potrebné dáta.

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.

Bez indexov

EXPLAIN (ANALYZE,BUFFERS) SELECT * FROM accounts WHERE followers_count < 10 AND friends_count > 1000 ORDER BY statuses_count;

1	Gather Merge (cost=203634.91..215791.48 rows=100530 width=118) (actual time=61282.861..61292.507 rows=719 loops=1)
2	[...] Workers Planned: 5
3	[...] Workers Launched: 5
4	[...] Buffers: shared hit=14650 read=157671 dirtied=2
5	[...] -> Sort (cost=202634.83..202685.10 rows=20106 width=118) (actual time=61199.869..61199.882 rows=120 loops=6)
6	[...] Sort Key: statuses_count
7	[...] Sort Method: quicksort Memory: 44kB
8	[...] Buffers: shared hit=14650 read=157671 dirtied=2
9	[...] Worker 0: Sort Method: quicksort Memory: 43kB
10	[...] Worker 1: Sort Method: quicksort Memory: 45kB
11	[...] Worker 2: Sort Method: quicksort Memory: 46kB
12	[...] Worker 3: Sort Method: quicksort Memory: 46kB
13	[...] Worker 4: Sort Method: quicksort Memory: 46kB
14	[...] -> Parallel Seq Scan on accounts (cost=0.00..201197.72 rows=20106 width=118) (actual time=796.045..61199.229 rows=12...
15	[...] Filter: ((followers_count < 10) AND (friends_count > 1000))
16	[...] Rows Removed by Filter: 1614421
17	[...] Buffers: shared hit=14465 read=157671 dirtied=2
18	Planning:
19	[...] Buffers: shared hit=21 read=2 dirtied=2
20	Planning Time: 2.789 ms
21	Execution Time: 61292.583 ms

V prípade bez indexu sa najprv vyhľadávajú záznamy ktoré vyhovujú podmienke WHERE. Potom sa vyhľadáva pomocou paralelného seq scanu.

Jakub Knežo Github repo = https://github.com/Kubislav/PDT-2-odovzdanie

S indexom

```
CREATE INDEX ON accounts(followers_count);
CREATE INDEX ON accounts(friends_count);
CREATE INDEX ON accounts(statuses_count);
EXPLAIN (ANALYZE,BUFFERS) SELECT * FROM accounts WHERE followers_count < 10 AND
friends_count > 1000 ORDER BY statuses_count;
```

	QUERY PLAN text
1	Gather Merge (cost=204611.22..216648.26 rows=100531 width=118) (actual time=981.639..999.036 rows=719 loops=1)
2	[...] Workers Planned: 4
3	[...] Workers Launched: 4
4	[...] -> Sort (cost=203611.16..203673.99 rows=25133 width=118) (actual time=908.425..908.439 rows=144 loops=5)
5	[...] Sort Key: statuses_count
6	[...] Sort Method: quicksort Memory: 49kB
7	[...] Worker 0: Sort Method: quicksort Memory: 45kB
8	[...] Worker 1: Sort Method: quicksort Memory: 52kB
9	[...] Worker 2: Sort Method: quicksort Memory: 47kB
10	[...] Worker 3: Sort Method: quicksort Memory: 52kB
11	[...] -> Parallel Bitmap Heap Scan on accounts (cost=26288.56..201774.27 rows=25133 width=118) (actual time=177.987..908.048 ro...
12	[...] Recheck Cond: ((followers_count < 10) AND (friends_count > 1000))
13	[...] Rows Removed by Index Recheck: 1150625
14	[...] Heap Blocks: exact=9609 lossy=20302
15	[...] -> BitmapAnd (cost=26288.56..26288.56 rows=100531 width=0) (actual time=227.677..227.678 rows=0 loops=1)
16	[...] -> Bitmap Index Scan on followers_count_btree_idx (cost=0.00..5726.33 rows=520253 width=0) (actual time=85.510..85.510 rows...
17	[...] Index Cond: (followers_count < 10)
18	[...] -> Bitmap Index Scan on friends_count_btree_idx (cost=0.00..20511.72 rows=1871904 width=0) (actual time=133.094..133.094 ro...
19	[...] Index Cond: (friends_count > 1000)
20	Planning Time: 4.094 ms
21	Execution Time: 999.152 ms

V prípade s indexom sa použije paralelný bitmap heap scan nad indexom pre followers_count pričom sa potom použije index condition, recheck condition a filter na atribút friends count. V oboch prípadoch sa data utrieďujú pomocou quicksrotu. Zo všetkých indexov malo zmysel použiť len index pre followers count keďže tento jediný sa aj využil.

Jakub Knežo Github repo = https://github.com/Kubislav/PDT-2-odovzdanie

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

```
DROP INDEX accounts_followers_idx;
DROP INDEX accounts_friends_idx;
DROP INDEX accounts_statuses_idx;
CREATE INDEX accounts_trio_idx ON accounts(followers_count, friends_count, statuses_count);

EXPLAIN (ANALYZE,BUFFERS) SELECT * FROM accounts
WHERE followers_count < 10 AND friends_count > 1000 ORDER BY statuses_count;
```

	QUERY PLAN text
1	Gather Merge (cost=190882.75..202919.79 rows=100531 width=118) (actual time=107.627..184.682 rows=719 loops=1)
2	[...] Workers Planned: 4
3	[...] Workers Launched: 4
4	[...] Buffers: shared hit=2307
5	[...] -> Sort (cost=189882.69..189945.52 rows=25133 width=118) (actual time=4.378..4.393 rows=144 loops=5)
6	[...] Sort Key: statuses_count
7	[...] Sort Method: quicksort Memory: 147kB
8	[...] Buffers: shared hit=2307
9	[...] Worker 0: Sort Method: quicksort Memory: 25kB
10	[...] Worker 1: Sort Method: quicksort Memory: 25kB
11	[...] Worker 2: Sort Method: quicksort Memory: 25kB
12	[...] Worker 3: Sort Method: quicksort Memory: 25kB
13	[...] -> Parallel Bitmap Heap Scan on accounts (cost=12560.10..188045.81 rows=25133 width=118) (actual time=3.975..4.232 rows=1...
14	[...] Recheck Cond: ((followers_count < 10) AND (friends_count > 1000))
15	[...] Heap Blocks: exact=718
16	[...] Buffers: shared hit=2279
17	[...] -> Bitmap Index Scan on accounts_trio_idx (cost=0.00..12534.96 rows=100531 width=0) (actual time=19.586..19.586 rows=719 l...
18	[...] Index Cond: ((followers_count < 10) AND (friends_count > 1000))
19	[...] Buffers: shared hit=1561
20	Planning:
21	[...] Buffers: shared hit=28 dirtied=3
22	Planning Time: 1.843 ms
23	Execution Time: 184.780 ms

V tomto prípade sa zachoval podobne ako v prípade s indexom.

Jakub Knežo Github repo = https://github.com/Kubislav/PDT-2-odovzdanie

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

```
EXPLAIN (ANALYZE,BUFFERS) SELECT * FROM accounts
WHERE followers_count < 1000 AND friends_count > 1000 ORDER BY statuses_count;
```

	QUERY PLAN text
1	Gather Merge (cost=242992.86..405577.48 rows=1344510 width=118) (actual time=1123.205..3492.593 rows=740655 loops=1)
2	[...] Workers Planned: 5
3	[...] Workers Launched: 5
4	[...] Buffers: shared hit=14686 read=157635, temp read=13537 written=13580
5	[...] -> Sort (cost=241992.78..242665.04 rows=268902 width=118) (actual time=1030.499..1079.002 rows=123443 loops=6)
6	[...] Sort Key: statuses_count
7	[...] Sort Method: external merge Disk: 20208kB
8	[...] Buffers: shared hit=14686 read=157635, temp read=13537 written=13580
9	[...] Worker 0: Sort Method: external merge Disk: 17728kB
10	[...] Worker 1: Sort Method: external merge Disk: 18384kB
11	[...] Worker 2: Sort Method: external merge Disk: 17112kB
12	[...] Worker 3: Sort Method: external merge Disk: 17120kB
13	[...] Worker 4: Sort Method: external merge Disk: 17744kB
14	[...] -> Parallel Seq Scan on accounts (cost=0.00..201197.73 rows=268902 width=118) (actual time=0.402..878.955 rows=123443 loo...
15	[...] Filter: ((followers_count < 1000) AND (friends_count > 1000))
16	[...] Rows Removed by Filter: 1491098
17	[...] Buffers: shared hit=14501 read=157635
18	Planning Time: 0.309 ms
19	Execution Time: 3769.869 ms

Použil sa paralelný sekvenčný scan. Nepoužili sa indexy keďže nájdených informácii je príliš veľa taktiež sa použil merge sort namiesto quicksortu. Tento merge sort sa kvôli veľkému množstvu nájdených informácii vykonával mimo RAMky.

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?

```
CREATE INDEX trigram_screen_name_idx ON accounts(screen_name)
CREATE INDEX b_tree_screen_name_idx ON accounts(screen_name)

EXPLAIN (ANALYZE,BUFFERS) SELECT * FROM accounts
WHERE screen_name LIKE 'realDonaldTrump'
```

Jakub Knežo Github repo = https://github.com/Kubislav/PDT-2-odovzdanie

	QUERY PLAN text
1	Index Scan using b_tree_screen_name_idx on accounts (cost=0.43..8.46 rows=1 width=118) (actual time=0.064..0.065 rows=1 loops=...
2	[...] Index Cond: ((screen_name)::text = 'realDonaldTrump'::text)
3	[...] Filter: ((screen_name)::text ~~ 'realDonaldTrump'::text)
4	[...] Buffers: shared hit=4
5	Planning Time: 0.157 ms
6	Execution Time: 0.082 ms

V prvom prípade kde sa hľadal presný výraz realDonaldTrump sa použil b_tree index keďže je rýchlejší ako sekvenčný scan a vyhľadáva sa presný výraz.

```
EXPLAIN (ANALYZE,BUFFERS) SELECT * FROM accounts
WHERE screen_name LIKE '%ldonaldt%'
```

	QUERY PLAN text
1	Bitmap Heap Scan on accounts (cost=127.51..3790.19 rows=969 width=118) (actual time=2.609..2.851 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	[...] Buffers: shared hit=76 read=78
6	[...] -> Bitmap Index Scan on trgm_idx (cost=0.00..127.27 rows=969 width=0) (actual time=2.550..2.550 rows=24 loops=1)
7	[...] Index Cond: ((screen_name)::text ~~ '%ldonaldt%'::text)
8	[...] Buffers: shared hit=76 read=54
9	Planning:
10	[...] Buffers: shared hit=13 read=11 dirtied=3
11	Planning Time: 5.725 ms
12	Execution Time: 2.883 ms

Pri vyhľadávaní pevnej časti textu tak je možné využiť b tree index. V prípade že hľadáme že string sa nachádza kdekoľvek je potrebné použiť trigram.

Jakub Knežo Github repo = https://github.com/Kubislav/PDT-2-odovzdanie

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.

```
CREATE EXTENSION btree_gin;
CREATE INDEX tweet_gin_content_idx ON tweets USING GIN(to_tsvector('english', content));
CREATE INDEX accounts_gin_description_idx ON accounts USING GIN(to_tsvector('english', description));
CREATE INDEX accounts_gin_name_idx ON accounts USING GIN(to_tsvector('english', name));

EXPLAIN (ANALYZE,BUFFERS)
SELECT  accounts.name, accounts.description, tweets.content, tweets.retweet_count
FROM tweets
JOIN accounts ON accounts.id = tweets.author_id
WHERE (to_tsvector('english', tweets.content) @@ to_tsquery('english', 'john & oliver'))
UNION ALL
SELECT accounts.name, accounts.description, tweets.content, tweets.retweet_count
FROM tweets
JOIN accounts ON accounts.id = tweets.author_id
WHERE (to_tsvector('english', accounts.name) @@ to_tsquery('english', 'john & oliver') OR
to_tsvector('english', accounts.description) @@ to_tsquery('english', 'john & oliver'))
ORDER BY retweet_count
```

	QUERY PLAN text
1	Sort (cost=1098313.21..1098319.21 rows=2397 width=250) (actual time=4492.328..4498.571 rows=603 loops=1)
2	[...] Sort Key: tweets.retweet_count
3	[...] Sort Method: quicksort Memory: 267kB
4	[...] Buffers: shared hit=5449 read=1024818
5	[...] -> Append (cost=58.63..1098178.66 rows=2397 width=250) (actual time=1.591..4497.953 rows=603 loops=1)
6	[...] Buffers: shared hit=5449 read=1024818
7	[...] -> Nested Loop (cost=58.63..10102.62 rows=799 width=250) (actual time=1.590..4.221 rows=452 loops=1)
8	[...] Buffers: shared hit=2249
9	[...] -> Bitmap Heap Scan on tweets (cost=58.20..3397.07 rows=799 width=171) (actual time=1.569..1.961 rows=452 loops=1)
10	[...] Recheck Cond: (to_tsvector('english'::regconfig, content) @@ "john" & "oliv"::tsquery)
11	[...] Heap Blocks: exact=393
12	[...] Buffers: shared hit=441
13	[...] -> Bitmap Index Scan on tweet_gin_content_idx (cost=0.00..58.00 rows=799 width=0) (actual time=1.528..1.528 rows=452 loops=1)
14	[...] Index Cond: (to_tsvector('english'::regconfig, content) @@ "john" & "oliv"::tsquery)
15	[...] Buffers: shared hit=48
16	[...] -> Index Scan using accounts_pkey on accounts (cost=0.43..8.39 rows=1 width=95) (actual time=0.004..0.004 rows=1 loops=452)
17	[...] Index Cond: (id = tweets.author_id)
18	[...] Buffers: shared hit=1808
19	[...] -> Gather (cost=3206.19..1088040.08 rows=1598 width=250) (actual time=4.732..4493.674 rows=151 loops=1)
20	[...] Workers Planned: 7
21	[...] Workers Launched: 7
22	[...] Buffers: shared hit=3200 read=1024818
23	[...] -> Hash Join (cost=2206.19..1086880.28 rows=228 width=250) (actual time=229.268..4366.324 rows=19 loops=8)
24	[...] Hash Cond: (tweets_1.author_id = accounts_1.id)

Jakub Knežo Github repo = https://github.com/Kubislav/PDT-2-odovzdanie

24	[...] Hash Cond: (tweets_1.author_id = accounts_1.id)
25	[...] Buffers: shared hit=3200 read=1024818
26	[...] -> Parallel Seq Scan on tweets tweets_1 (cost=0.00..1072683.07 rows=4568007 width=171) (actual time=0.400..3482.823 rows=3997006 loops=8)
27	[...] Buffers: shared hit=2185 read=1024818
28	[...] -> Hash (cost=2200.14..2200.14 rows=484 width=95) (actual time=1.591..1.593 rows=43 loops=8)
29	[...] Buckets: 1024 Batches: 1 Memory Usage: 15kB
30	[...] Buffers: shared hit=672
31	[...] -> Bitmap Heap Scan on accounts accounts_1 (cost=91.87..2200.14 rows=484 width=95) (actual time=1.374..1.560 rows=43 loops=8)
32	[...] Recheck Cond: ((to_tsvector('english'::regconfig, (name)::text) @@ "john" & "oliv"::tsquery) OR (to_tsvector('english'::regconfig, description) @@ "john" & "oliv"::tsquery))
33	[...] Heap Blocks: exact=43
34	[...] Buffers: shared hit=672
35	[...] -> BitmapOr (cost=91.87..91.87 rows=484 width=0) (actual time=1.359..1.360 rows=0 loops=8)
36	[...] Buffers: shared hit=328
37	[...] -> Bitmap Index Scan on accounts_gin_name_idx (cost=0.00..45.82 rows=242 width=0) (actual time=1.028..1.028 rows=18 loops=8)
38	[...] Index Cond: (to_tsvector('english'::regconfig, (name)::text) @@ "john" & "oliv"::tsquery)
39	[...] Buffers: shared hit=208
40	[...] -> Bitmap Index Scan on accounts_gin_description_idx (cost=0.00..45.82 rows=242 width=0) (actual time=0.329..0.329 rows=25 loops=8)
41	[...] Index Cond: (to_tsvector('english'::regconfig, description) @@ "john" & "oliv"::tsquery)
42	[...] Buffers: shared hit=120
43	Planning:
44	[...] Buffers: shared hit=21
45	Planning Time: 0.571 ms
46	Execution Time: 4498.804 ms

Pri OR podmienke musíme prechádzať obidve posting listy čo je časovo náročnejšie aj keď prejdeme kratší posting list musíme ešte skontrolovať ten druhý ktorý je dlhší. Pri AND podmienke prechádzame posting listy rychlejšie pričom skončíme prechádzanie keď skončíme kratší z dvoch posting listov.