

Elektronski fakultet Niš
Sistemi za upravljanje bazama podataka

Interna struktura i organizacija indeksa PostgreSQL baze podataka

Seminarski rad

Mentor:
Doc. dr Aleksandar Stanimirović

Student:
Julije Kostov 1026

Sadržaj

1.Uvod.....	3
2.Indeksi.....	5
2.1 Izgled datoteke tabele	5
2.2 Kako rade indeksi	6
2.3 PostgreSQL indeksi	7
2.4 Implementacija indeksa	8
3.Primer Korišćenja indeksa	13
4.Zaključak	21
Literatura.....	22

1. Uvod

Koncept baze podataka nastao je početkom 1970-ih godina kao sistem koji se koristi za organizovanje podataka u kolekciju kojoj se lako pristupa i brzo pretražuje. Baze podataka rešavaju problem upravljanja podacima, obezbeđuju pouzdanost, integritet prilikom vršenja operacija nad podacima i nezavisnost podataka od softvera.

Indeksi kod baza podataka predstavljaju strukturu podataka koja se koristi za ubrzanje pretraživanja podataka, a za uzvrat zauzimaju dodatan prostor na disku i zahtevaju dodatne operacije za održavanje te strukture. Indeksi se kreiraju nad kolonom određene tabele kako bi se ubrzalo pretraživanje podataka iz te kolone. Pored toga moguće je kreirati i indekse nad više kolona ili parcijalne indekse koji se kreiraju samo za određene redove.

Indeksi se koristi kako bi se sekvencijalna pretraga podataka u bazi zamenila bržim načinom jer je kod velikih baza podataka sekvencijalna pretraga podataka veoma spora. Kako bi ubrzali pretragu podataka indeksi koriste različite strukture podataka i u zavisnosti od same strukture zavisi ubrzanje pretrage, memorija koju oni zauzimaju na disku i dodatne operacije za održavanje te strukture. Osim za pretraživanje podataka indeksi se mogu koristiti kako bi se implemetirala ograničenja nad kolonom baze kao što su UNIQUE, EXCLUSION, PRIMARY KEY i FOREIGN KEY.

Postoje različite metode indeksiranja:

- Neklasterizovani – podaci su prisutni u proizvoljnom redosledu, ali logički poredak određuje indeks. Neklasterizovano stablo indeksa sadrži sortirane indeksirane ključeve sa listovima koji pokazuju na red u bazi.
- Klasterizovani – fizički podaci su poređani po redosledu kao u indeksu, pa time je moguće kreiranje samo jednom klasterizovanog indeksa nad tabelom. Pošto su redovi poređani jedan iza drugoga u određenim slučajevima potrebno je manje operacija za čitanje.

Postoje različiti tipovi indeksa:

- Bitmape – za čuvanje podataka koriste niz bitova i prilikom upita rade se operacije na bitovima
- Gusti indeksi – koristi datoteku koja se sastoji od parova ključ i pokazivač na podatak u datoteci sa podacima.
- Jedinstevni indeksi – koristi datoteku koja se sastoji od parova ključ i pokazivač na blok u datoteci sa podacima.
- Obrnuti indeksi – pre unosa vrednosti indeksa u datoteku okrenu vrednost ključa. Primer 1234 se unosi kao 4321. Najčešće se koriste kod sekvenci gde se nova vrednost dobija povećanje zadnje.

- Primarni indeks – koristi se kod kolone koja predstavlja primarni ključ kolone
- Sekundarni indeksi – koriste se kod kolona koje se ne koriste za određivanje redosleda niti su ključevi.

PostgreSQL ili Postgres predstavlja jedan od najpopularnijih sistema za upravljanje bazama podataka koji je otvorenog koda. To znači da je svakome dozvoljeno korišćenje i modifikacija softvera za svoje potrebe. Osim besplatne verzije postoji i enterprise verzija obogaćena alatima koja je namenjena velikim sistemima.

PostgreSQL je objektno-relaciona baza podataka otvorenog koda napisana na programskom jeziku C. Nastala je nadogradnjom projekta POSTGRES koji je započet još 1986 a koristio se za analizu i skladištenje različitih tipova podataka. Dolazi sa velikim brojem ugrađenih alata koji olakšavaju rad sa bazom bez obzira na količinu podataka koja se skladišti. Ima ACID svojstva i podržava veliki deo SQL standarda: kompleksno pretraživanje, strani ključevi, okidači (eng. *trigger*), pogledi (eng. *view*) koji se mogu ažurirati, integritet transakcije, itd. Pored toga omogućava korisnicima nadogradnju funkcionalnosti i bez ponovnog generisanja izvršnog fajla baze [1]. Moguće je dodati nove tipove podataka, funkcije, operatore, funkcije za agregaciju, načine za indeksiranje, itd.

Karakteristike PostgreSQL baze podataka:

- Podrška za veliki broj različitih tipova podataka
- Integritet podataka
- Konkurentnost
- Pouzdanost i oporavak od greške
- Sigurnost
- Proširljivost

Pored toga što je Postgres relaciona baza podataka, ima i određene karakteristike objektno baze. To znači da ona ima objektno-orijentisane karakteristike kao što su: koncept nasleđivanja, mogućnost definisanja kompleksnih tipova podataka i funkcija za rad sa njima. Većina korisnika ne koristi ove funkcionalnosti ali u pojedinim slučajevima mogu da pomognu.

Drugo poglavlje rada opisuje indekse PostgreSQL baze podataka, prednosti i mane njihovog korišćenja, način kako oni funkcionišu i strukture podatka koje se koriste za njihovu implementaciju.

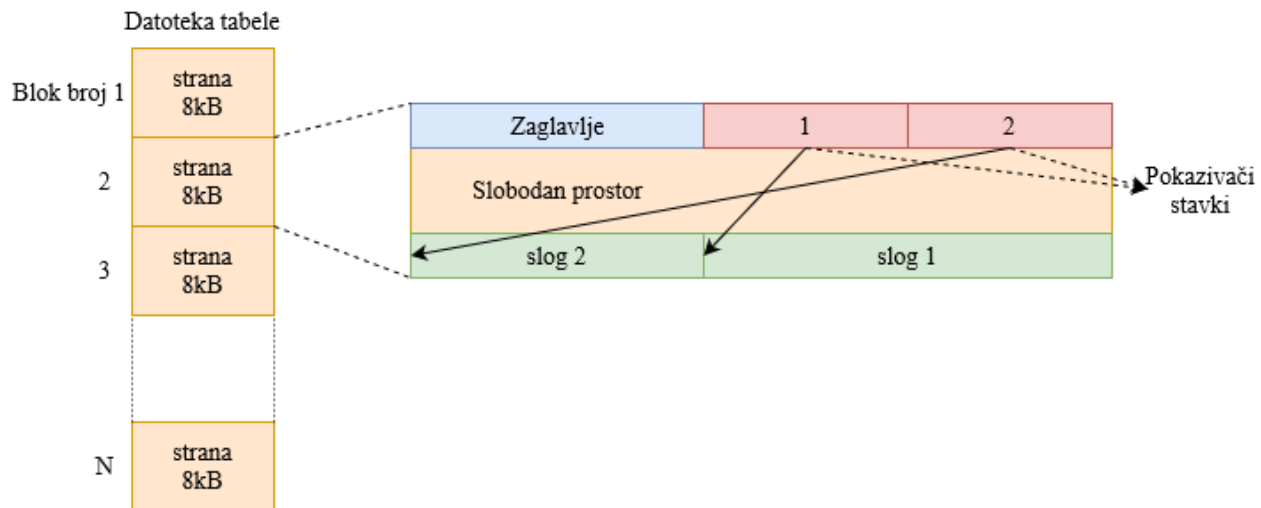
U trećem poglavlju rada dati su praktični primeri korišćenja indeksa kao i performanse izvršenja upita sa i bez njihovog korišćenja.

2. Indeksi

Indeksi u bazama podataka predstavljaju strukturu podataka koja se koristi kako bi se ubrzale određene operacije nad podacima koji se čuvaju u bazi. Korišćenjem indeksa dobija se ubrzanje ali cena korišćenja je povećanje veličine podataka i dodatne operacije koje se izvršavaju prilikom upisa novih podataka radi održanja strukture indeksa. Prilikom pretrage podataka koristi se metoda sekvencijalnog pretraživanja podataka i to je veoma skupo, korišćenjem indeksa izbegava se ovaj način traženja. Indekse je moguće kreirati za jednu ili više kolona tabele, a najbolje je kreirati indekse za one kolone koje se često pretražuju.

2.1 Izgled datoteke tabele

Kako bi razumeli kako rade indeksi baze podataka potrebno je znati kako baza podataka skladišti podatke na disku. Svaka datoteka sa podacima je podeljena na strane (blokove) fiksne dužine čija je dužina obično 8kB. Te strane su numerisane tako da svaka strana ima svoj broj i on predstavlja broj bloka. Kada se datoteka napuni Postgres dodaje novu praznu stranu na kraj datoteke kako bi joj se povećala veličina. Unutrašnji izgled datoteke podataka zavisi od tipa objekta za koji je ta datoteka namenjena [2]. Na slici 2.1 prikazana je struktura heap datoteke tabele.



Slika 2.1 Izgled heap datoteke tabele

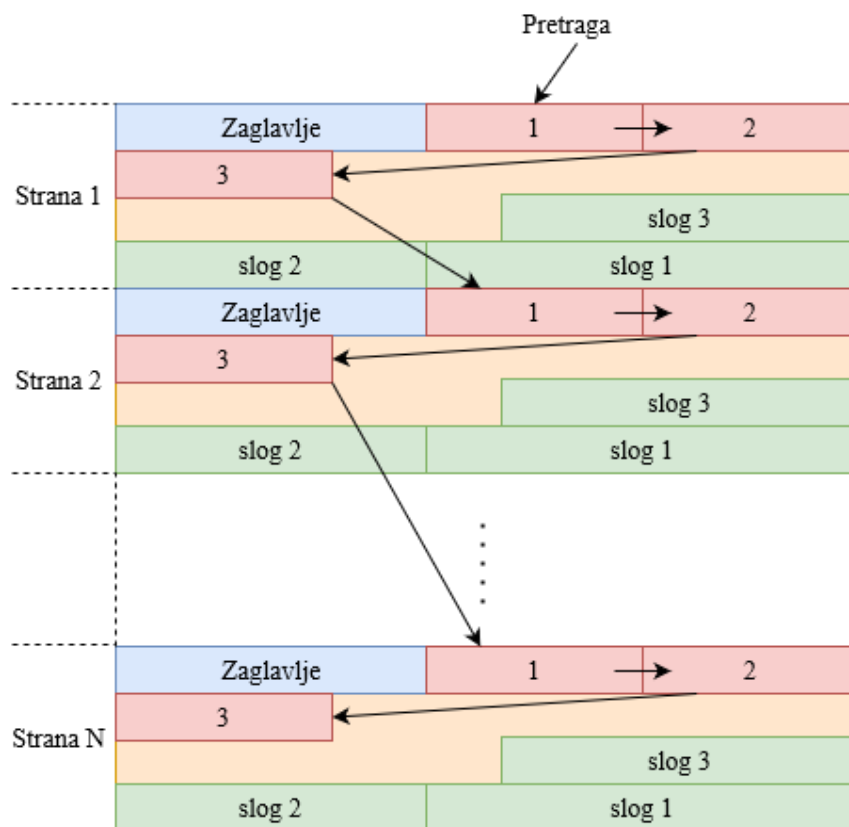
Podaci koji se nalaza u okviru datoteke tabele:

- Slogovi podataka – Predstavljaju same podatke
- Pokazivači stavki – Prestavlja pokazivač veličine 4B koji pokazuje na slog i zove se pokazivač stavke. Pokazivači su uređeni u niz i počinju sa indeksom 1. Posle dodavanja novog sloga dodaje se i novi pokazivač u niz.

- **Zaglavlje** – Postoji na početku svake strane i sadrži osnovne informacije o strani, dužine je 24B. Sastoji se od polja `pd_lsn`, `pd_checksum`, `pd_lower`, `pd_upper`, `pd_special`, `pd_pagesize_version` i `pd_prune_xid`. `Pd_lower` polje pokazuje na zadnji pokazivač, a `pd_upper` na prvi najbliži slog.

2.2 Kako rade indeksi

Na slici 2.1 videli smo kako se tabela čuva u datoteku koja predstavlja niz strana. Prilikom pretrage podataka potrebno je proveriti da li svaka strana zadovoljava kriterijum pretrage. Svaki upit koji se izvršava u bazi ima određenu cenu koja je direktno povezana sa brojem strana koje se učitavaju i kojima se pristupa. Baze podataka mogu da rade i bez indeksa ali problem je brzina. Ako za kolonu po kojoj se vrši pretraga ne postoje indeksi tada se vrši sekvencijalna pretraga i prolazi se kroz sve strane i podatke [3]. Na slici 2.2 prikazana je sekvencijalna pretraga podataka.



Slika 2.2 Sekvencijalna pretraga podataka

Kako bi se izbegla sekvencijalna pretraga podataka moguće je definisati indekse nad kolonom. Indeksi u suštini čuvaju ključ-vrednost podatke, gde ključ predstavlja vrednost iz kolone, a vrednost je par broj strane i broj sloga. Cena indeksa koja se mora platiti je dodatna memorija za skladištenje ovih podataka. Pored toga indeksi se moraju održavati, tako da

prilikom unosa ažuriranja ili brisanja podataka indeksi se takođe moraju ažurirati što zahteva dodatne operacije u zavisnosti od tipa indeksa. Indekse je moguće definisati za više kolona i samo za određene redove koji ispunjavaju neki uslov, takvi indeksi se nazivaju delimični (eng. *partial*).

2.3 PostgreSQL indeksi

Postoje različiti tipovi indeksa, PostgreSQL podržava sledeće: B-stablo, hash, GiST, SP-GiST, GIN i BRIN. Svaki od ovih tipova koristi različiti algoritam koji je optimizovan za određene pretrage. Indeksi koji se kreiraju komandom CREATE INDEX su tipa B-stablo i oni se najčešće koriste [2].

B-stabla

Postgres uključuje implementaciju standardnog B-stabla koje predstavlja višesmerno balansirano stablo. Moguće ih je koristiti za sve podatke koji mogu da se sortiraju u linearni niz. B-stabla se koriste od strane PostgreSQL planera upita kada se vrši pretraga po indeksiranoj koloni sa jednim od sledećih operatora: <, <=, =, >= i >. Takođe B-stabla se koriste i sa operatorima BETWEEN, IN, IS NULL i IS NOT NULL. B-stable se mogu koristiti i sa operatorom LIKE u slučaju da je početak stringa poznat, npr. LIKE 'tes%' .

Hash

Hash indeksi se koriste samo sa konkretnim poređenjem. Planer upita koristi hash indekse samo kada se vrši pretraga po indeksiranoj koloni sa operatorom =. Komanda za kreiranje indeksa je:

```
CREATE INDEX <ime_indeksa> ON <tabela> USING HASH (<kolona>)
```

GiST

GiST je skraćenica za eng. *Generalized Search Tree* što znači generalizovano stablo pretrage. Kod njih su definisane metode koje se koriste za pristup strukturi stabla, a konkretnu implementaciju te strukture je moguće menjati, pa i definisati svoje tipove podataka sa tim metodama. GiST indeksi nisu jedinstvenog tipa i za njih mogu da se koriste različite strategije. Operatori sa kojima se GiST indeksi mogu koristiti zavise od strategije koja se koristi. Primer operatora za dvodimenzionalne geometrijske podatke su: <<, &<, &>, >>, <<|, &<|, |&>, |>>, @>, <@, ~= I &&. Takođe oni mogu da optimizuju pronalaženje najbližeg suseda u upitima za zadatu tačku.

SP-GiST

SP-GiST je skraćenica za eng. *space-partitioned* GiST što znači podeljeni prostor. Oni koriste podeljena stabla pretrage koja imaju za cilj da izmapiraju čvorove stabla na stranice na

disku tako da se prilikom pretrage smanji broj pristupa stranama. Kao i GiST indeksi oni pružaju infrastrukturu za različite tipove pretraga. Moguće ih je koristiti sa različitim strukturama podataka kao što su: quad stabla, k-d stabla i radix stabla.

GIN

GIN je skraćenica za eng. *Generalized inverted index*. GIN indeksi predstavljaju invertovane indekse koji se koriste za tipove podataka koji sadrže više vrednosti, kao što su nizovi. Invertovani indeksi sadrži jedan unos za svaku vrednost i veoma je efikasan kod pretraga gde se proverava da li postoji vrednost u nizu. Kao i kod GiST i SP-GiST indeksa i GIN indeksi podržavaju više strategija za indeksiranje pa se tako operatori sa kojima oni koriste razlikuju u zavisnosti od izabrane strategije.

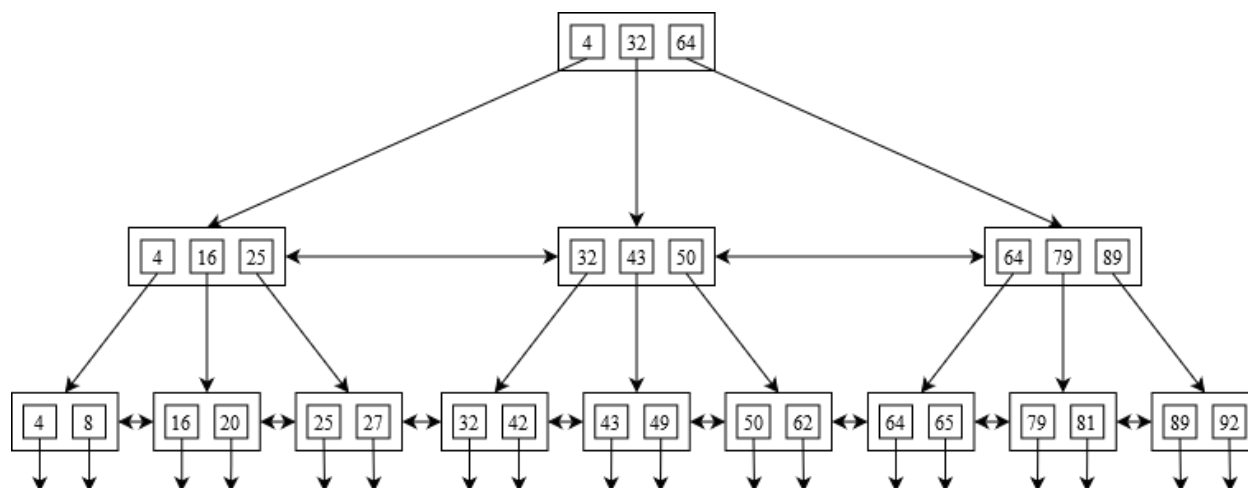
BRIN

BRIN je skraćenica za eng. *Block range index*. Namenjeni su za velike tabele kod kojih određene kolone imaju vezu sa fizičkom lokacijom u tabeli. Blok u određenom opsegu predstavlja grupu strana koje su fizički susedne u tabeli. Brin indeksi čuvaju određene sumariizovane podatke o vrednostima koje se nalaze u strani. Primer predstavlja minimalna i maksimalna vrednost u okviru strane. Kako je i ovde moguće koristiti različite strategije, operatori sa kojima se oni koriste zavise od same strategije.

2.4 Implementacija indeksa

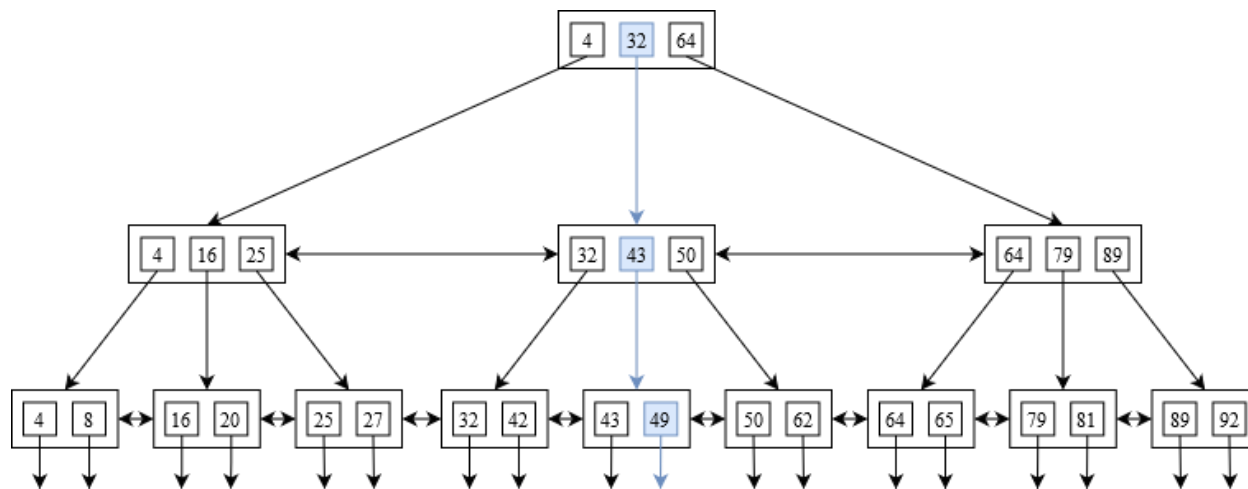
B-stabla

Koriste se kod podataka koji mogu da se sortiraju. B-stabla predstavljaju balansirana stabla i na njihovim listovima nalaze se podaci indeksa koji pokazuju na redove u tabeli. Kod balansiranih stabala svaki list je podjedano udaljen od korena stabla. Zbog toga pretraga bilo koje vrednosti traje isto. Listovi su povezani tako da svaki list ima pokazivač na sledeći i naredni čvor, pa nije potrebno vraćati se na koren svaki put prilikom preuzimanja poređanih podataka. Kod unutrašnjih čvorova svaki čvor pokazuje na dete koje sadrži vrednosti iz roditelja i ona je minimalna za taj čvor. Na slici 2.3 prikazan je primer B-stabla.



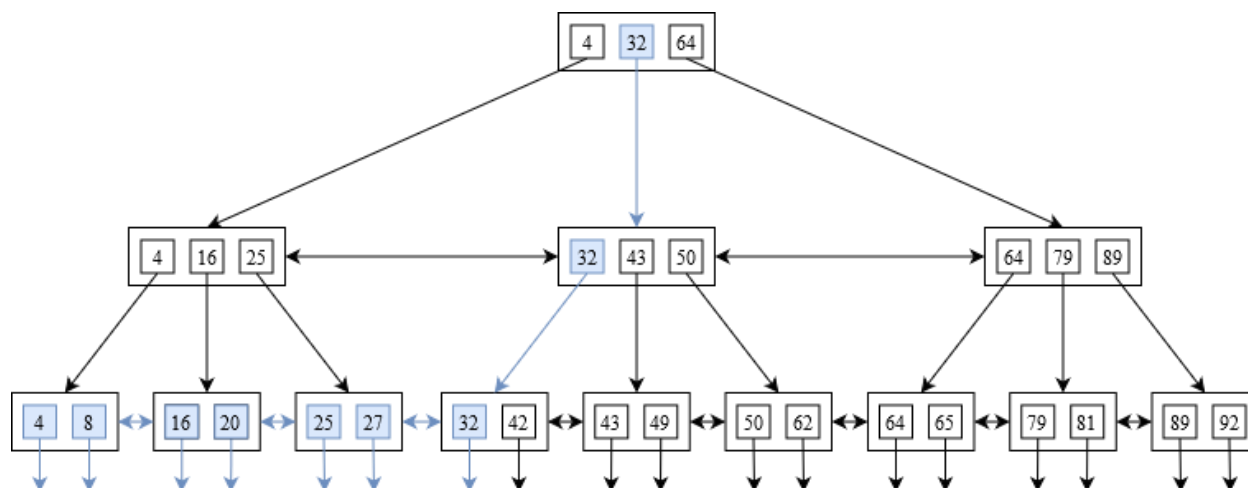
Slika 2.3 Primer B-stabla

Ako je potrebno izvršiti pretragu podataka tako da je tražena vrednost 49, postupak je sledeći: kreće se od vrha stabla i za svaki čvor se traži naredni dok se ne stigne do lista. U svakom kako bi se našao naredni ispituje se tražena vrednost sa vrednostima unutar čvora. Pošto je $32 < 49 < 64$ uzima se sledeći čvor na kog pokazuje vrednost 32. Ovaj postupak se radi rekursivno sve dok se ne stigne do lista. List sadrži indeks koji pokazuje na broj strane i broj sloga koji sadrži traženi podatak i on se uzima. Na slici 2.4 prikazan je primer obilaska stabla za broj 49.



Slika 2.4 Obilazak stabla za pretragu broj 49

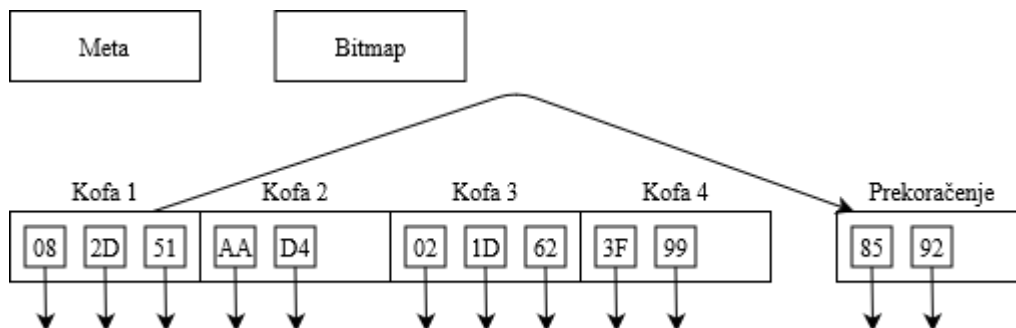
Na slici 2.5 prikazan je obilazak stabla za slučaj pretrage vrednosti manje od 35.



Slika 2.5 Obilazak stabla za pretragu vrednosti manje od 35

Hash indeksi

U mnogim programskim jezicima postoji tip heš tablica. Postgres koristi sličnu implementaciju heš tablica. Heš tablice predstavljaju niz od N vrednosti koje su indeksirane heširanom vrednošću a njihova vrednost pokazuje na par broj strane i broj sloga. Heš vrednost dobija se korišćenjem heš funkcije. Ova funkcija mora dobro da distribuira vrednosti i da se izvršava brzo. Problem je što se nekad dešava da se za različitu vrednost dobije isti heš. Zbog toga heš tablica je organizovana u kofe (eng. *bucket*) koje služe za čuvanje pokazivača na različite redove u tabeli koji imaju isti heš. Na slici 2.6 prikazan je primer heš tablice.



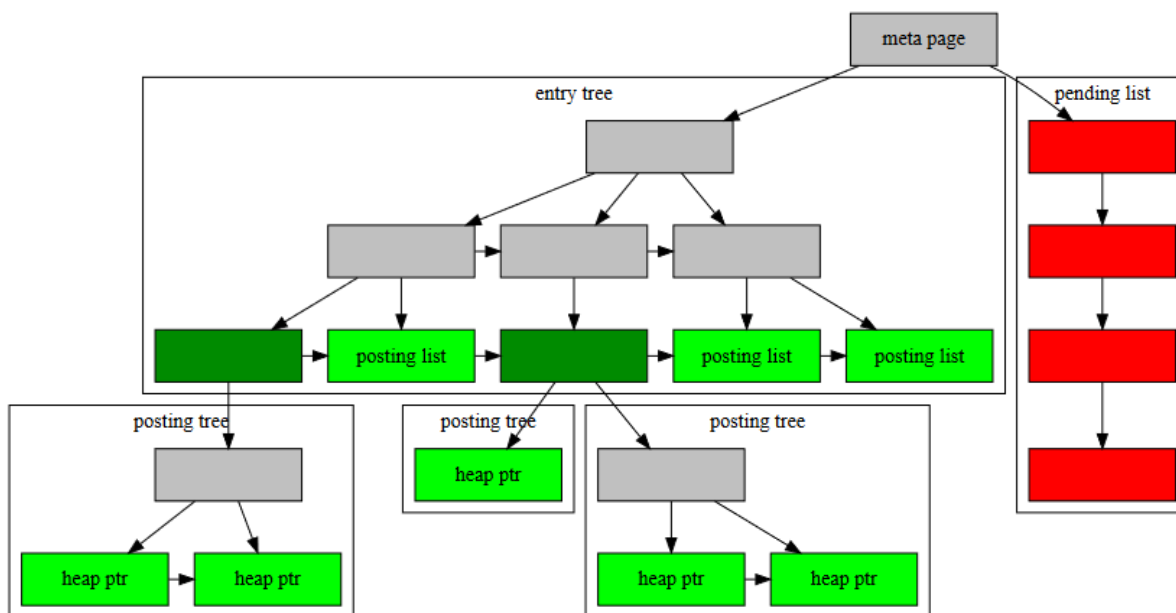
Slika 2.6 Haš tablica

Kofa za prekoračenje se koristi kada jedna kofa nije dovoljna za sve vrednosti. Bitmapa se koristi kako bi se pratilo koje kofe za prekoračenje su prazne i mogu se koristiti za druge kofe. Heš tablice funkcionišu tako što se za određenu vrednost odradi heš funkcija. Dobija vrednost je broj kofe gde je potrebno smesiti indeksiranu vrednost. Kada se vrši pretraga po ključu potrebno je izvršiti haš funkciju na vrednost gde se dobija broj kofe a nakon toga potrebno je proveriti sadržaj kofe i vratiti samo onu vrednost koja je tražena [2].

GIN indeksi

GIN indeksi se koriste kod slučajeva gde stavke koje se indeksiraju predstavljaju kompozitne vrednosti, a upiti koji se optimizuju pomoću ovih indeksa traže jedan element koji se nalazi u okviru kompozitne vrednosti. Na primer, moguće je indeksiranje više dokumenata, a upiti koji koriste ove indekse vrše pretragu dokumenata koji sadrži određenu reč. Kod GIN indeksa reč *stavka* koristimo za kompozitne vrednosti koje se indeksiraju, a reč ključ za vrednost elementa.

GIN indeksa čuva par eng. *key*, *posting list*, gde je *posting list* niz identifikatora redova koji sadrže ključ. Isti identifikator nekog reda može da se pojavi u više *posting list*-i. Svaka vrednost ključa se čuva samo jednom, pa su GIN indeksi efikasni za slučajeve gde se ključ pojavljuje više puta. GIN indeksi mogu da sadrže B-stablo definisano za odgovarajući ključ, gde je svaki ključ deo jedne ili više indeksiranih stavki i gde svaki slog u listu sadrži pokazivač na B-stablo indeksa ili jednostavnu listu indeksa [2]. Na slici 2.7 prikazane su komponente GIN indeksa.



Slika 2.7 Komponente GIN indeksa

GiST

GiST predstavlja balansirano stablo kod koga listovi sadrže neki predikat i pokazivač na red u tabeli. Taj predikat obično predstavlja neki boolean izraz, a indeksirani podaci moraju da zadovoljavaju taj predikat. Svaki unutrašnji čvor sadrži predikat i pokazivač na decu. Sva deca određenog čvora moraju da zadovoljavaju i predikat roditelja. Za pretraživanje GiST indeksa koristi se posebna funkcija. U zavisnosti i podataka koji se indeksiraju kao i struktura podataka koje se koriste za implementaciju indeksa ta funkcija se razlikuje. Ona mora da ima određeni interfejs koji se implementira. Ova funkcija se poziva za indeksirane podatke i proverava da li predikat u indeksu odgovara predikatu pretraživanja. Za interne čvorove u stablu ova funkcija

određuje da li je potrebno ići dalje na decu čvora, dok za listove određuje da li podaci ulaze u pretragu. Pretraživanje kreće od korena stabla i proverava koje podstablo možda ispunjava uslove i onda se za svako podstablo izvršava isti algoritam dok se ne dođe do listova koji ispunjavaju uslov.

3. Primer korišćenja indeksa

Poređenje performansi indeksa rađeno je na udaljenoj virtuelnoj mašini koja ima 1 CPU od 1.8 GHz sa 30720kB keša, 1GB RAM memorije i 25GB SSD. Na toj virtuelnoj mašini podignuta je PostgreSQL baza u okviru Docker kontejnera, a naredba za to je:

```
docker run -d --restart unless-stopped - POSTGRES_USER=root -e POSTGRES_PASSWORD=dmbstest1232 -e POSTGRES_DB=dbms -p 5000:5432 postgres:12.2-alpine
```

Na podignutoj bazi kreirane su dve tabele *users* i *posts*, naredba za kreiranje table je:

```
CREATE TABLE users (  
    id SERIAL PRIMARY KEY,  
    name TEXT NOT NULL,  
    created_at TIMESTAMP NOT NULL DEFAULT NOW()  
);  
  
CREATE TABLE posts (  
    id SERIAL PRIMARY KEY,  
    user_id INTEGER NOT NULL REFERENCES users (id),  
    public BOOLEAN NOT NULL DEFAULT TRUE,  
    text TEXT NOT NULL,  
    created_at TIMESTAMP NOT NULL DEFAULT NOW()  
);
```

Indeksi se automatski kreirani samo za kolone *id* jer su one primarni ključ. Posle kreiranja table potrebno je dodati podatke, što je učinjeno komandama:

```
INSERT INTO users (name, created_at)  
SELECT  
    ARRAY_TO_STRING(ARRAY(SELECT SUBSTR('abcdefghijklmnopqrstuvxyz', ((RANDOM()*(30-1)+1)::INTEGER),1) FROM GENERATE_SERIES(1,7 + b * 0)), ''),  
    NOW() + (RANDOM() * (NOW() + '365 days' - NOW()))  
FROM GENERATE_SERIES(1, 5000) as a(b);  
  
INSERT INTO posts (user_id, text, public, created_at)  
SELECT  
    RANDOM() * 4999 + 1,  
    ARRAY_TO_STRING(ARRAY(SELECT SUBSTR('abcdef ghi jklmno pqrst uvwxyz', ((RANDOM()*(34-1)+1)::INTEGER),1) FROM GENERATE_SERIES(1,120 + b * 0)), ''),  
    CASE WHEN RANDOM() > 0.3 THEN TRUE ELSE FALSE END,  
    NOW() + (RANDOM() * (NOW() + '365 days' - NOW()))  
FROM GENERATE_SERIES(1, 500000) as a(b);
```

U tabeli *users* dodato je 5000 redova, veličina datoteke koju ova tabela zauzima na disku je 256kB. Na slici 3.1 prikazan je primer umetnutih podataka u tabelu *users*.

```
1 SELECT * FROM users;
```

Data Output	Explain	Messages	Notifications
	id [PK] integer	name text	created_at timestamp without time zone
1	1	flzzj	2020-05-11 16:33:58.594603
2	2	pjnmy...	2020-06-26 14:32:09.164618
3	3	fqzyqrz	2021-03-14 02:12:44.204916
4	4	kjwiyv	2020-08-29 21:43:36.534963
5	5	lkgu	2021-01-17 12:15:48.639957
6	6	xwoqyz	2020-03-30 05:57:01.663116
7	7	snnzqm	2020-05-28 22:23:28.093653
8	8	zxrkshj	2021-02-06 16:55:55.318386
9	9	rqzyue	2020-06-13 22:09:41.579976
10	10	weqtia	2020-10-13 05:17:00.259743

Slika 3.1 Deo umetnutih podataka u tabelu *users*

U tabeli *posts* dodato je 500000 redova, veličina datoteke koju ova tabela zauzima na disku je 78MB. Na slici 3.2 prikazan je primer umetnutih podataka u tabelu *posts*.

```
1 SELECT * FROM posts;
```

Data Output		Explain	Messages	Notifications		
	id [PK] integer	user_id integer	public boolean	text text	created_at timestamp without time zone	
1		1	4830	true	qnk wx...	2020-07-22 08:51:04.159649
2		2	4900	false	t l nuk ...	2021-02-18 02:13:19.862107
3		3	2191	false	kg bpn...	2020-10-11 07:50:53.892299
4		4	3268	true	k ufvp...	2020-09-10 06:59:28.795342
5		5	3894	false	utwzr o...	2020-05-06 23:22:27.014382
6		6	957	true	wux w...	2020-09-01 23:39:02.881987
7		7	712	false	gtluu s...	2020-05-28 12:04:46.460573
8		8	3807	true	m qfzt...	2021-02-24 04:05:10.100405
9		9	2514	true	ahh c...	2020-10-26 11:49:51.191283
10		10	1608	true	bkrun k...	2021-01-25 16:54:36.705054

Slika 3.2 Deo umetnutih podataka u tabelu *posts*

Na slici 3.3 predstavljen je upit za preuzimanje podataka iz *users* tabele po imenu.

```
1 SELECT * FROM users WHERE name = 'fbxjnl';
```

	Data Output	Explain	Messages	Notifications
	id [PK] integer	name text	created_at timestamp without time zone	
1	305	fbxjnl	2020-11-03 18:10:17.115782	

Slika 3.3 Upit za preuzimanje korisnika po imenu

Kako bi proverili cenu izvršenje ovog upita ispred komande potrebno je dodati naredbe EXPLAIN ANALYZE. Na slici 3.4 prikazan je plan i cena izvršenja prethodnog upita.

```
1 EXPLAIN ANALYZE SELECT * FROM users WHERE name = 'fbxjnl';
```

	Data Output	Explain	Messages	Notifications
	QUERY PLAN text			
1	Seq Scan on users (cost=0.00..94.50 rows=1 width=19) (actual time=0.078..0.767 rows=1 loops=1)			
2	Filter: (name = 'fbxjnl'::text)			
3	Rows Removed by Filter: 4999			
4	Planning Time: 0.089 ms			
5	Execution Time: 0.802 ms			


Slika 3.4 Plan i cena izvršenja prethodnog upita

Sa slike 3.4 vidi se da kako bi se pronašao korisnik sa specificiranim imenom baza vrši sekvencijalnu pretragu podataka, što znači da prolazi kroz sve podatke i vrši filtriranje svih redova osim jednog. Pored toga vidi se da je cena izvršenja upita 94.50, a vreme izvršenja 0.802ms. Komandom:

```
CREATE INDEX index_users_name ON users USING HASH (name);
```

kreiraćemo indekse za kolonu *name* tabele *users*. Nakon dodavanja indeksa za kolonu *name* kreirana je datoteka na disku koja zauzima 272kB i sadrži indekse za tu kolonu. Na slici 3.5 prikazan je plan izvršenja i cena.

```
1 EXPLAIN ANALYZE SELECT * FROM users WHERE name = 'fbxjnl';
```


Data Output	Explain	Messages	Notifications
QUERY PLAN  text			
1	Index Scan using index_users_name on users (cost=0.00..8.02 rows=1 width=19) (actual time=0.028..0.029 rows=1 loops=1)		
2	Index Cond: (name = 'fbxjnl'::text)		
3	Planning Time: 0.082 ms		
4	Execution Time: 0.060 ms		

Slika 3.5 Plan i cena izvršenja prethodnog upita sa indeksima

Sa slike 3.5 vidi se da se sada ne radi sekvencijalna pretraga podataka već se vrši pretraga indeksa što dovodi do toga da je cena upita 8.02, a vreme izvršenja upita 0.060ms što je više od 10 puta brže od slučaja bez indeksa. Izvršenje upita jeste brže i cena je manja ali dodata je datoteka koja sadrži indekse i njena veličina je u ovom slučaju 272kB dok je veličina indeksa za kolonu *id* 128kB. Sada prilikom svakom ažuriranja tabele potrebno je ažurirati i ove dve datoteke sa indeksima. Ako se često vrši pretraga podataka po koloni *name* ovo predstavlja malu cenu koju treba platiti.

Na slici 3.6 prikazan je upit i plan njegovog izvršenja za preuzimanje podataka iz tabele *posts* za korisnika sa određenim identifikatorom.

```
1 EXPLAIN ANALYZE SELECT * from posts WHERE user_id = 4831;
```

Data Output	Explain	Messages	Notifications
QUERY PLAN  text			
1	Gather (cost=1000.00..13569.17 rows=100 width=128) (actual time=0.566..96.495 rows=104 loops=1)		
2	Workers Planned: 2		
3	Workers Launched: 2		
4	-> Parallel Seq Scan on posts (cost=0.00..12559.17 rows=42 width=128) (actual time=3.208..81.111 rows=35 loops=3)		
5	Filter: (user_id = 4831)		
6	Rows Removed by Filter: 166632		
7	Planning Time: 0.088 ms		
8	Execution Time: 96.565 ms		

Slika 3.6 Plan i cena izvršenja upita za preuzimanje postova određenog korisnika

Tabela *posts* ima 500000 redova i zbog toga se vidi da se u planu za pretraživanje podataka puštaju 2 radnika koja paralelno pretražuju sve podatke tabele. Cena izvršenja ovog upita je 12559.17, a vreme izvršenja 96.565ms. Komandom:

```
CREATE INDEX index_posts_user_id ON posts (user_id);
```

kreiraju se indeksi za kolonu *user_id*. Veličina datoteke koja sadrži indekse za ovu kolonu je 10MB. Radi poređenja, veličina indeksa za kolonu *id* je 11MB, što znači da indeksi nad ovom tablom zauzimaju 21MB, a sama veličina tabele je 78MB. Na slici 3.7 prikazan je plan i cena izvršenja indeksa nakon kreiranja indeksa.

```
1 EXPLAIN ANALYZE SELECT * FROM posts WHERE user_id = 4381;
```

Data Output	Explain	Messages	Notifications
QUERY PLAN			
text			
1	Bitmap Heap Scan on posts (cost=5.20..376.38 rows=100 width=128) (actual time=0.063..0.577 rows=104 loops=1)		
2	Recheck Cond: (user_id = 4381)		
3	Heap Blocks: exact=103		
4	-> Bitmap Index Scan on index_posts_user_id (cost=0.00..5.17 rows=100 width=0) (actual time=0.042..0.042 rows=104 loops=1)		
5	Index Cond: (user_id = 4381)		
6	Planning Time: 0.284 ms		
7	Execution Time: 0.617 ms		

Slika 3.7 Plan i cena izvršenja prethodnog upita sa indeksima

Sa slike 3.7 vidi se da je cena izvršenja ovog upita 376.38, dok je vreme izvršenja 0.617ms.

Osim indeksa za jednu kolonu moguće je kreirati i indekse za više kolona. Takvi indeksi se koriste za upite gde postoji bilo koji podskup indeksiranih kolona, međutim on je najefikasniji ako je prisutna kolona koja se nalazi na prvom mestu u indeksu. Ovo označava da je redosled nabiranja kolona u indeksu bitan i efikasnost upita zavisi od korišćenja kolona u njemu. Pored indeksa nad više kolona moguće je kreirati i parcijalne indekse. Kod njih se definiše indeks samo za određene podatke. Primer parcijalnog indeksa:

```
CREATE INDEX index_posts_user_id_public ON posts (user_id) WHERE public = TRUE;
```

Ovom komandom kreira se indeks nad kolonom *user_id* samo za one postove koji su javni. Na slici 3.8 prikazan je upit gde se pretražuju postovi za određenog korisnika.

```
1 EXPLAIN ANALYZE SELECT* FROM posts WHERE user_id = 3821;
```

Data Output	Explain	Messages	Notifications
QUERY PLAN text			
1	Gather (cost=1000.00..13569.17 rows=100 width=128) (actual time=1.721..94.041 rows=87 loops=1)		
2	Workers Planned: 2		
3	Workers Launched: 2		
4	-> Parallel Seq Scan on posts (cost=0.00..12559.17 rows=42 width=128) (actual time=0.367..81.145 rows=29 loops=3)		
5	Filter: (user_id = 3821)		
6	Rows Removed by Filter: 166638		
7	Planning Time: 0.444 ms		
8	Execution Time: 94.095 ms		

Slika 3.8 Pretraga postova za određenog korisnika

Sa slike 3.8 vidi se da se ne koristi prethodno kreirani indeks, već da se vrši sekvencijalna pretraga svih podataka. Cena izvršenja upita je 13569.17, a vreme izvršenja 94.095ms. Na slici 3.9 prikazan je upit gde se traže javni postovi za određenog korisnika.

```
1 EXPLAIN ANALYZE SELECT* FROM posts WHERE user_id = 3821 AND public = TRUE;
```

Data Output	Explain	Messages	Notifications
QUERY PLAN text			
1	Bitmap Heap Scan on posts (cost=4.97..268.23 rows=70 width=128) (actual time=0.045..0.356 rows=64 loops=1)		
2	Recheck Cond: ((user_id = 3821) AND public)		
3	Heap Blocks: exact=64		
4	-> Bitmap Index Scan on index_posts_public (cost=0.00..4.95 rows=70 width=0) (actual time=0.029..0.029 rows=64 loops=1)		
5	Index Cond: (user_id = 3821)		
6	Planning Time: 0.124 ms		
7	Execution Time: 0.384 ms		

Slika 3.9 Pretraga javnih postova za određenog korisnika

Sa slike 3.9 vidi se da se u ovom slučaju koristi prethodno kreirani parcijalni indeks. Sada je cena ovog upita 268.23, a vreme izvršenja 0.384. Veličina datoteke koja sadrži parcijalni indeks za kolonu *user_id* je 7MB, dok je veličina indeksa nad svim podacima ove kolone 10MB. Ukoliko imamo dosta upita sa određenim filtriranjima moguće je ta filtriranja izdvojiti i u

indeksu kako bi se smanjila veličina indeksa. Problem je što izostavljanjem dela za filtriranje se indeksi ne koriste za izvršenje upita.

Kako bi se kreirao GIN indeks potrebno je da kolona za koju se kreira bude tipa *ts_vector*. Kreirana baza podataka nema tip podataka niz. Umesto toga koristićemo *gin_trgm_ops* funkciju koja prilikom kreiranja indeksa za kolonu kreira trigrame. Trigram je struktura podataka koja čuva 3 slova reči. Kako bi se koristila ova funkcija potrebno je omogućiti *pg_trgm* ekstenziju što se radi sledećom komandom:

```
CREATE EXTENSION pg_trgm;
```

Nakon izvršenja prethodne komande sledećom komandom kreiraćemo GIN indeks nad kolonom *text* tabele *posts*:

```
CREATE INDEX index_posts_text ON posts USING gin(text gin_trgm_ops);
```

Veličina kreiranog indeksa je 235MB, dok je veličina tabele 78MB. Na slici 3.10 prikazan je plan izvršenja upita koji vrši pretraživanje teksta bez korišćenja indeksa, a na slici 3.11 prikazan je plan izvršenja istog upita samo korišćenjem GIN indeksa.

1	EXPLAIN ANALYZE SELECT * from posts WHERE text LIKE '%ato%';
Data Output Explain Messages Notifications	
	QUERY PLAN
	text
1	Gather (cost=1000.00..13564.17 rows=50 width=128) (actual time=0.527..282.427 rows=1016 loops=1)
2	Workers Planned: 2
3	Workers Launched: 2
4	-> Parallel Seq Scan on posts (cost=0.00..12559.17 rows=21 width=128) (actual time=0.135..261.189 rows=339 loops=3)
5	Filter: (text ~~ '%ato% '::text)
6	Rows Removed by Filter: 166328
7	Planning Time: 0.177 ms
8	Execution Time: 282.570 ms

Slika 3.10 Plan izvršenja upita gde se pretražuje *text* kolona *posts* tabele

1	EXPLAIN ANALYZE SELECT * from posts WHERE text LIKE '%ato%';
<div> Data Output Explain Messages Notifications </div>	
	QUERY PLAN text
1	Bitmap Heap Scan on posts (cost=20.39..210.38 rows=50 width=128) (actual time=0.437..2.305 rows=1016 loops=1)
2	Recheck Cond: (text ~~ '%ato% '::text)
3	Heap Blocks: exact=976
4	-> Bitmap Index Scan on index_posts_text (cost=0.00..20.38 rows=50 width=0) (actual time=0.260..0.260 rows=1016 loops=1)
5	Index Cond: (text ~~ '%ato% '::text)
6	Planning Time: 0.151 ms
7	Execution Time: 2.410 ms

Slika 3.11 Plan izvršenja prethodnog upita korišćenjem GIN indeksa

Sa slike 3.10 vidi se da je cena upita 13564.17 jer se kreiraju dva radnika koja vrše sekvencijalnu pretragu podataka, a izvršenje upita traje 282.570ms. Sa slike 3.11 vidi se da je cena izvršenja upita 210.38 jer se za pretragu koristi kreirani GIN indeks, a izvršenje upita traje 2.410ms. Korišćenjem indeksa dobili smo ubrzanje veće od 100 puta, ali veličina datoteke koja sadrži indekse je 235MB.

4. Zaključak

U ovom radu opisani su koncepti, implementacija i korišćenje određenih tipova indeksa za PostgreSQL bazu podataka. Baza podataka za svoje funkcionisanje ne zahteva postojanje indeksa, ali njene performanse su drastično slabije bez postojanja istih. Cena koju moramo platiti prilikom kreiranja indeksa jeste dodatna memorija i operacija prilikom modifikacije podataka radi održavanja indeksa. U zavisnosti od količine podataka i broja upita nad određenim podacima potrebno je izvršiti analizu i pažljivo iskoristiti indekse.

LITERATURA

- [1] „*Postres about*“ - <https://www.postgresql.org/about/>, Poslednji pristup: 02.04.2020.
- [2] „*PostgreSQL 12.2 Documentation*” - <https://www.postgresql.org/files/documentation/pdf/12/postgresql-12-A4.pdf>, Poslednji pristup: 02.04.2020.
- [3] „*Indexes in PostgreSQL*“ - <https://postgrespro.com/blog/pgsql/3994098>, Poslednji pristup: 02.04.2020.