Seminarski rad

Indeksi u PostgreSQL

Sistemi baza podataka

Stefan Mladenov 1205

# 

[1. Uvod 3](#_Toc132743837)

[2. Indeksi u bazama podataka 4](#_Toc132743838)

[3. Indeksi u PostgreSQL 6](#_Toc132743839)

[3.1 Kreiranje i brisanje indeksa 6](#_Toc132743840)

[3.2 B-Tree 8](#_Toc132743841)

[3.2.1 Indeksi nad jednom kolonom 10](#_Toc132743842)

[3.2.2 Indeksi nad više kolona (engl. *multi-column*) 10](#_Toc132743843)

[3.2.3 Parcijalni indeksi (engl. *partial*) 12](#_Toc132743844)

[3.3 Hash 13](#_Toc132743845)

[3.3.1 Poređenje B-tree i Hash 15](#_Toc132743846)

[3.4 GiST 18](#_Toc132743847)

[3.5 SP-GiST 19](#_Toc132743848)

[3.6 GIN 19](#_Toc132743849)

[3.7 BRIN 21](#_Toc132743850)

[4. Zaključak 24](#_Toc132743851)

[5. Reference 25](#_Toc132743852)

# 

# 

# 1. Uvod

Danas, u doba digitalizacije i automatizacije procesa, baze podataka predstavljaju neizostavni deo sistema koji se bave određenom obradom informacija, koje je potrebno skladištiti kako bi se iznova koristile. Količina podataka koji se skladište raste, pa samim tim i vreme potrebno za čitanje i pronalaženje željenih podataka. Stoga se javila potreba za optimizacijom, gde je jedan od načina koji donosi potrebno ubrzanje - indeksiranje.

U realnom svetu, puno je primera koji oslikavaju pojam indeksiranja. Nekada, kada nije bilo mobilnih telefona, brojevi telefona su se zapisivali u telefonskim imenicima. Kako je broj upisanih kontakta rastao, tako je raslo i vreme potrebno za pronalaženje željenog broja telefona, što je rezultiralo pojavom organizacije istih, gde su neki od imenika sadržali i slova uz pomoć kojih je bila olakšana navigacija do strana koje sadrže brojeve telefona kontakta koji počinju zadatim slovom.

Knjige takođe sadrže vid indeksiranja, gde je kroz sadržaj moguće naći željenu oblast u kraćem roku, bez potrebe da se ide kroz svaku stranu knjige.

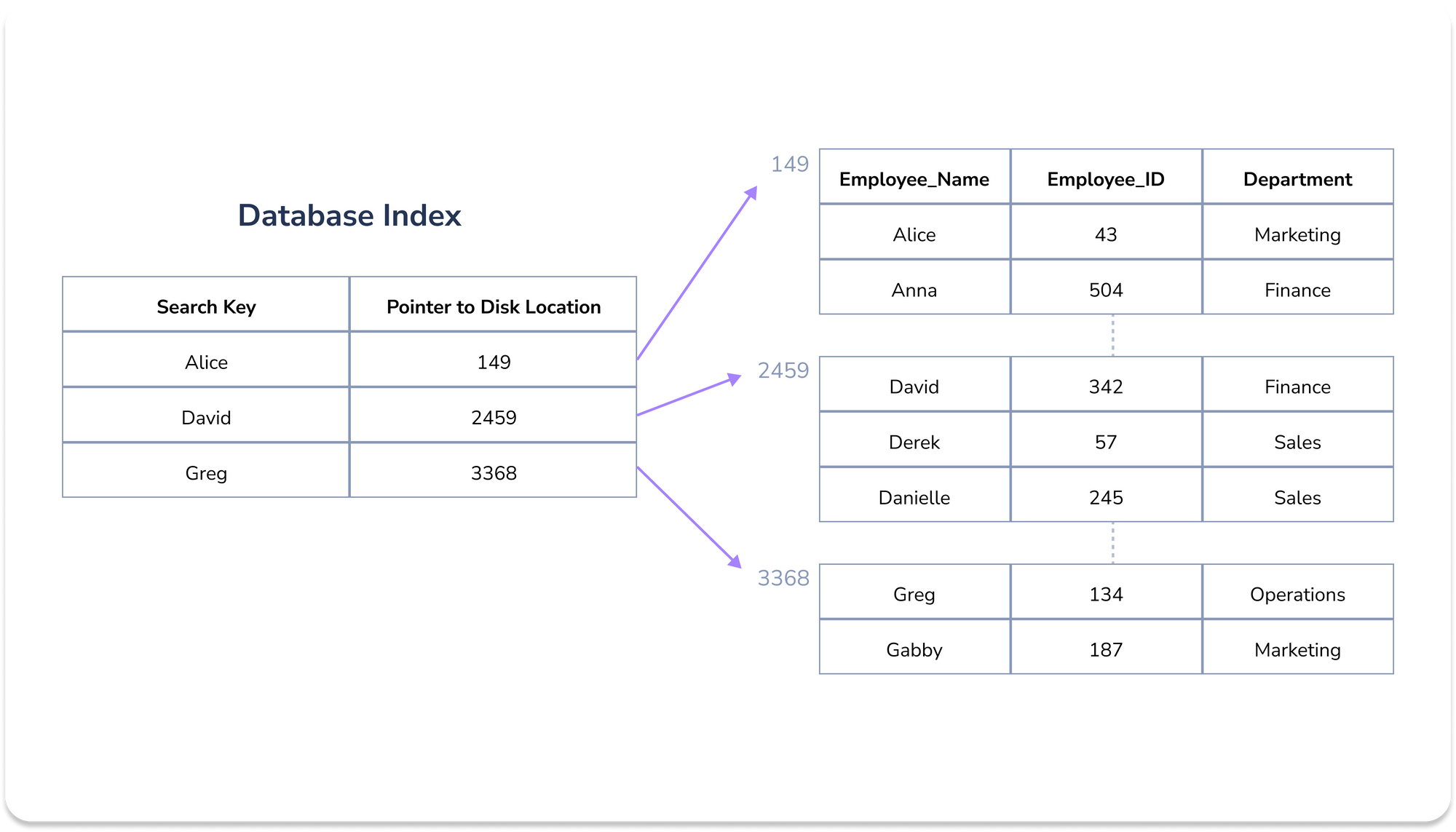


Slika 1. Telefonski imenik

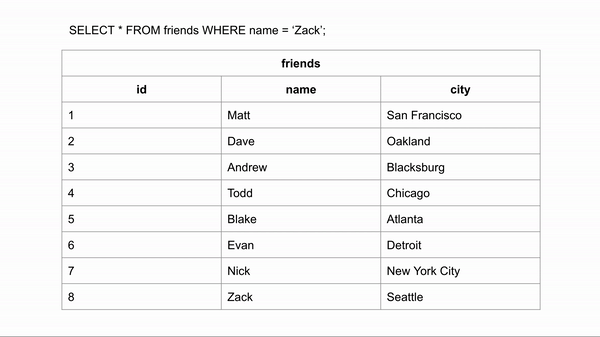
# 2. Indeksi u bazama podataka

Baze podataka se mogu posmatrati kao knjige. Podaci koji se dodaju se pamte u okviru blokova ili stranica (u zavisnosti od DB *engine*-a se zovu drugačije, ali upućuju na isti pojam). Kada je jedna stranica puna, novi podaci se dodaju na novoj strani.

Indeksi baze podatka predstavljaju strukturu podataka koja sadrži referencu na određeni red ili redove u bazi, kreiranoj nad jednom ili više kolona. Predstavljaju svojevrsne *lookup* tabele uz pomoć kojih se traženi podaci brže lociraju, pa su samim tim i određeni upiti brži.



Slika 2. Indeks - *pointer*

Na ovaj način se izbegava najsporija opcija pretrage, bez indeksa, a to je pretraga svih redova sekvencijalno dok se ne nadje poklapanje.  


Slika 3. *Full-table* scan

Sa druge strane, operacija upisa novih redova u tabeli, nad kojom postoje indeksi, je duža i sama operacija je skuplja jer je potrebno ažurirati i indeks tabelu, što može prouzrokovati i izvršenje nekih dodatnih akcija poput rekalkulacije i repozicioniranja čvorova u grafu(stablu), što zahteva i procesorsko vreme, tako da je potrebno naći pravi balans i ne praviti indekse po svaku cenu. Između ostalog, te indeksne strukture zauzimaju dodatan prostor na samom fizičkom memorijskom medijumu, kao i prostor u okviru memorije radi obrade.

Za samo održavanje te indeksne strukture podataka je zadužen *engine* baze podataka.

# 

# 3. Indeksi u PostgreSQL

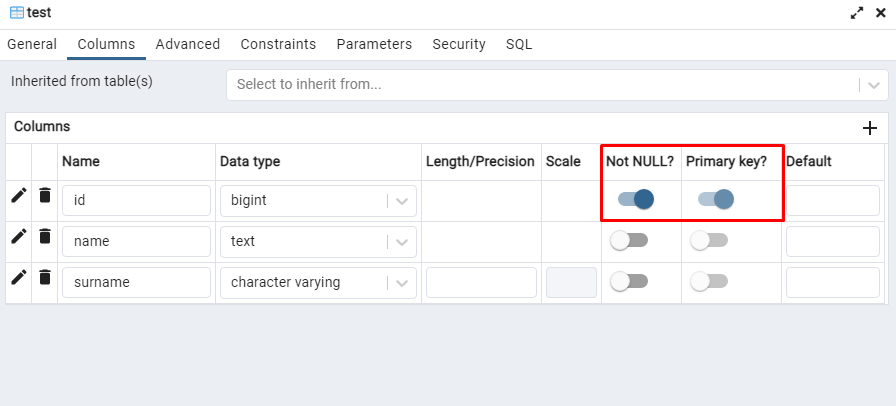
PostgreSQL je popularna *open-source* objektno-relaciona baza podataka koja pored osnovnih svojstava relacionih baza, ima odličnu podršku za integraciju sa različitim objektno-orijentisanim jezicima. Odlikuje je brzina, sigurnost, pouzdanost, sposobnost manipulacije velikom količinom podataka, proširljivost, fleksibilnost, kao i širok dijapazon funkcionalnosti, između ostalih i mogućnost implementacije različitih tipova indeksa:

* B-tree
* Hash
* GiST
* SP-GiST
* GIN
* BRIN

gde svaki sa svojim karakteristikama, koristeći drugačiji algoritam, nalazi primenu u rešavanju problema sa performansama, ubrzavajući čitanje i pretragu redova u tabeli.

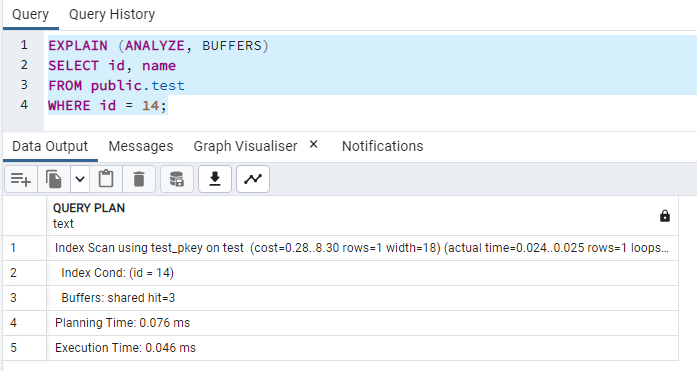
Indekse je moguće kreirati nad jednom (engl.*single-column*) ili više kolona(engl. *multi-column*), kao i parcijalne (engl. *partial*) i jedinstvene (engl. *unique*). Pored prethodno pomenutih, postoje i implicitni indeksi koji su kreirani od strane samog *engine*-a.

## 3.1 Kreiranje i brisanje indeksa

Kada definišemo primarni ključ za određenu tabelu, ona će po automatizmu dodati unique indeks za polje koje predstavlja primarni ključ:  


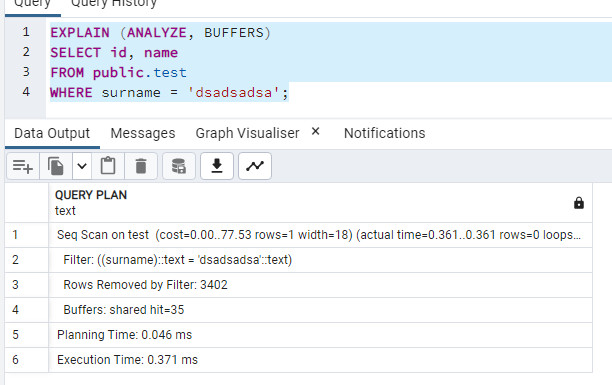
Slika 4. Primarni ključ

Ukoliko pokušamo da pročitamo neki rekord za konkretan id:



Slika 5. Pretraga po indeksu

Dok slučaju pretrage tabele po vrednosti kolone koja nije indeksirana, poput *surname* u konkretnom primeru, te je moguće uočiti razliku u vremenu izvršenja:



Slika 6: Pretraga po koloni bez indeksa

Ukoliko želimo da kreiramo indeks, to je moguće uraditi uz pomoć komande: CREATE INDEX *ime\_indeksa* ON *ime\_tabele*(*kolona*);

Reverzna akcija, tj. uklanjanje indeksa moguće je korišćenjem komande:

DROP INDEX *ime\_indeksa*;

Komandom za kreiranje indeksa se po default-u kreira B-tree indeks koji nije unique i nije klasterovan. Za upotrebu drugih tipova indeksa, potrebno je navesti ime tipa:

CREATE INDEX *ime\_indeksa* ON *ime\_tabele* USING *tip\_indeksa*(*kolona*);

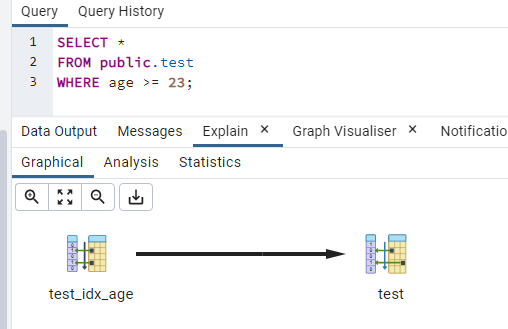
DB engine će po automatizmu ažurirati indekse kada dođe do promene i uključiti ih u planiranje upita. Ažuriranje i brisanje podataka takođe može biti brže korišćenjem indeksa, ukoliko se koriste u samom uslovu i povezivanju (engl. *JOIN*). Dok sa druge strane, kreiranje indeksa nad velikim tabelama može trajati dugo. Predefinisano ponašanje ove baze u slučaju istovremene operacije čitanja i kreiranja indeksa je moguće, za razliku od operacija upisa i brisanja koje su blokirane dok se kreacija indeksa ne završi. Ovo je moguće prevazići i omogućiti paralelno izvršenje upisa i kreiranja indeksa.

Indekse je moguće kreirati i nad kolonama koje je potrebno transformisati nekom funkcijom, što je naročito korisno prilikom čestih upita koje sadrže transformacije nad vrednostima kolona poput datuma:

CREATE INDEX *ime\_indeksa* ON *ime\_tabele(funkcija(parametar))*

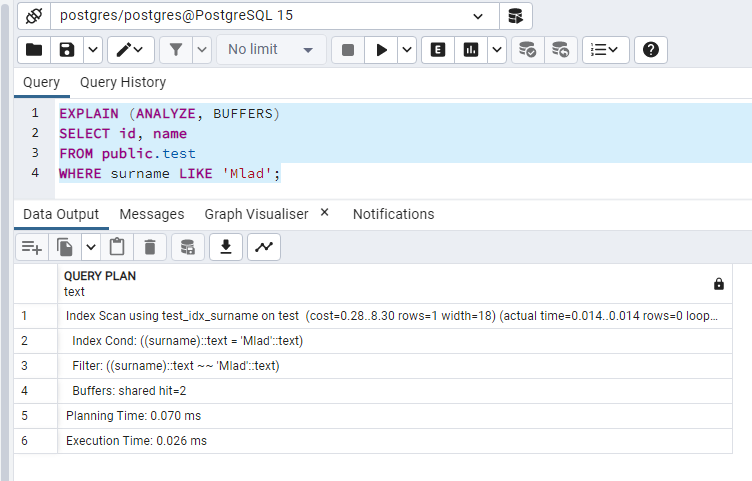
## 3.2 B-Tree

Ovaj tip indeksa je default-ni indeks kod PostgreSQL-a i biće uključen u plan izvršenja upita kod operacija poređenja i uslova kojim se određuje neki opseg ( ‘<’, ‘<=’, ‘=’, ‘>=’, ‘>’). Koriste se i u slučaju korišćenja operatora poput BETWEEN i IN, koji takođe određuju opseg.



Slika 7. Korišćenje indeksa kod upita sa opsegom

Takođe, kod provere vrednosti - *IS NULL* i *IS NOT NULL*. Biće uključen u plan izvršenja upita i prilikom korišćenja *LIKE*.



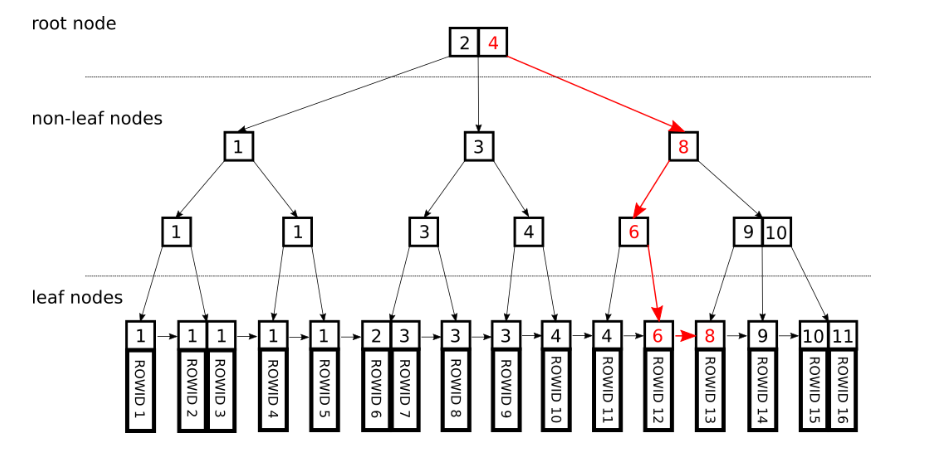
Slika 7. LIKE operator

B-tree je sortirano izbalansirano stablo koje održava hijerarhiju(organizaciju) velike količine podataka koje omogućava manipulaciju čvorova. Kompleksnost je logaritamska kada se govori u kontekstu upisa, brisanja i pretrage. Za razliku od binarnog stabla, čvorovi kod B-tree stabla mogu imati veći broj čvorova “dece”.

Generalno, svako traženje prema zadatom indeksu se odvija u tri koraka:

* Prolazak kroz stablo sve do dolaska do listova stabla
* Praćenje lanca čvorova kroz koji se prolazi sve dok se ne nađu svi listovi koji zadovoljavaju zadati uslov
* Pribavljanje podataka sa diska

### 3.2.1 Indeksi nad jednom kolonom



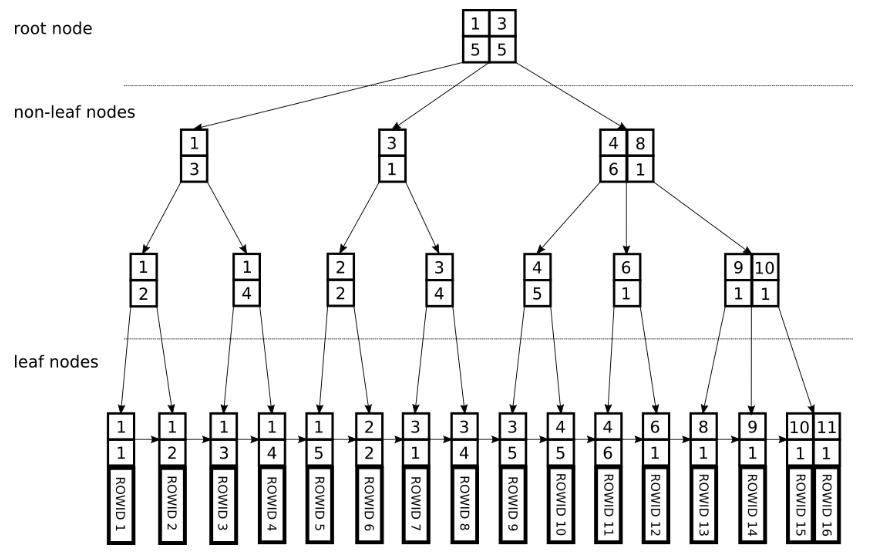
Slika 8. Indeks *lookup* - indeksi nad jednom kolonom

Na prethodnoj slici je moguće videti put kojim se ide prilikom pretrage koja se vrši po ključu čija je vrednost 8. Stablo je sortirano u rastućem redosledu i obilazi se s leva na desno. Na prvom koraku pretrage, ide se na krajnji desni krak (strelica 4 ka 8), s obzirom na to da čvorovi koji se nalaze na najvišem nivou, takozvani čvorovi ‘koreni’ (engl. *root node*), sadrže ključeve koji su manji od tražene vrednosti. Nakon toga, nailaskom na čvor sa traženim ključem na nivou koji nije krajnji, a ni list (engl. *internal node*), navigira se na levu, manju vrednost čime se pristupa listovima stabla. Nakon toga se kroz lančanu listu elemenata na poslednjem nivou dolazi do lista sa traženim ključem, nakon čega sledi pristup memorijskom medijumu sa željenim podacima preko ROWID-ja.

### 3.2.2 Indeksi nad više kolona (engl. *multi-column*)

Pored kreiranja indeksa nad jednom kolonom, moguće je kreirati i indekse u okviru kojih bi se sagledavale vrednosti većeg broja kolona i na osnovu toga kreiralo stablo. PostgreSQL omogućava kreaciju indeksa nad maksimalno 32 kolone, gde ključevi ne predstavljaju jedinstvene vrednosti, već se nalaze u formi kolekcije ili struktura sa većim brojem komponenti (engl. *tuple*). Prilikom definisanja kolona koje će biti uključene u sam indeks, potrebno je voditi računa o granularnosti samih podataka, tj. potrebno je uključiti kolone sa manjom granularnošću, što praktično znači da je potrebno naći kolone čije vrednosti variraju najmanje, kako bi se *recordi* podelili u manji broj grupa, koje sadrže veći broj podataka. Naravno, ne treba se striktno držati ovog principa, jer ukoliko se podaci u okviru upita ne filtriraju/zahtevaju po tom parametru koji ima najmanju granularnost, već po nekom drugom i grupišu se po njemu, onda je bolje izvršiti podelu po tom parametru. Treba voditi računa o redosledu kojim se navode imena samih kolona, jer će kolone koje su prve navedene biti kolone lideri, po njima će se prvi izvršiti podela i imaće “najveći prioritet”. Takođe treba uzeti u obzir i obilazak listova za pronalazak željenih podataka, što je manje skokova između listova, to će rezultati optimizacije biti bolji.

Jedan od primera iz realnog sveta bi bio telefonski imenik, gde su podaci sortirani u abecednom redosledu. Sortiranje se vrši na osnovu, uglavnom, na osnovu početnog slova nečijeg prezimena. Kako broj osoba sa istim prezimenom raste, tako pretraga postaje otežana, pa je zapravo najbolje napraviti indekse na osnovu prezimena i imena, gde bi prezime bilo prvonavedeno, što bi značilo da se traženje prvo vrši po prezimenu, pa se onda pronalaskom prezimena, traženje vrši po imenu, pri čemu su kontakti sa istim prezimenom sortirani po imenu, takođe u abecednom redosledu.



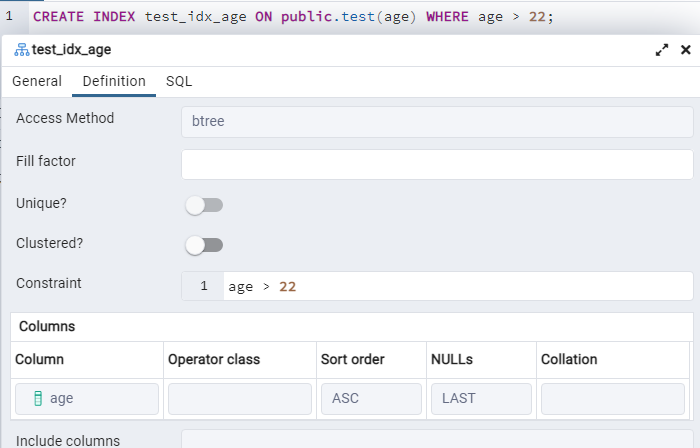
Slika 9. Indeksi nad više kolona(engl. *multi-column*)

### 3.2.3 Parcijalni indeksi (engl. *partial*)

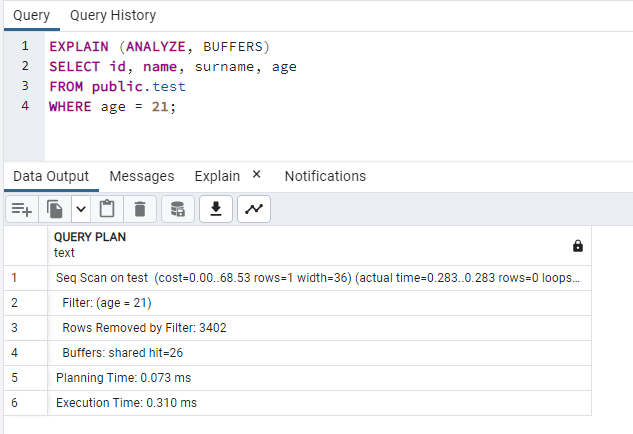
Parcijalni indeksi predstavljaju indekse koje je moguće kreirati ne samo na nivou kolone, već na nivou reda, definisanjem uslova koji svaki red mora da ispunjava. Na taj način se tabela deli na manje grupe, gde upiti koji pokušavaju pribavljanje redova iz nekog manjeg dela tabele su brži, dok u slučaju pribavljanje veće količine podataka, koji se nalaze u okviru redova koji nisu obuhvaćeni indeksom (jer ne ispunjavaju indeksnu klauzulu), pretraga po indeksu se svodi na sekvencijalnu pretragu tabele. Za razliku od standardnog indeksa, ovaj indeks zauzima manje memorije i sama pretraga je brža kao i operacije upisa, jer se indeks ne ažurira u svim slučajevima.

Ove indekse je moguće kreirati:

CREATE INDEX *ime\_indeksa* ON *ime\_tabele*(*kolona*) WHERE …



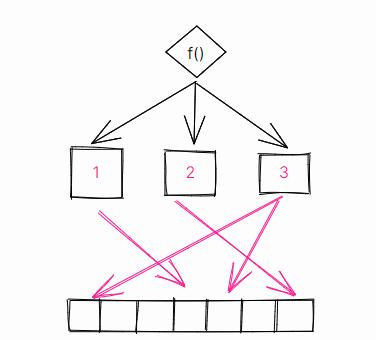
Slika 10. Kreiranje parcijalnog indeksa



Slika 11. Traženje rekorda sekvencijalno sa parcijalnim indeksom nad age kolonom

## 3.3 Hash

Hash indeksi koriste *hash* strukturu podataka. Na osnovu *hash* funkcije, kreira se 32-bitni kod (moguće generisati oko 4 milijardi kombinacija).



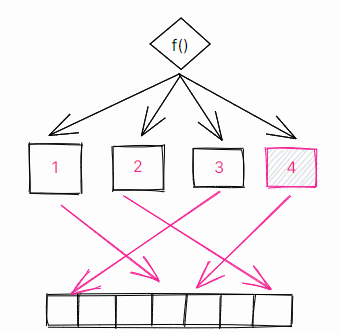
Slika 12. Hash indeks

Kada se doda nova vrednost u tabelu, izvršava se *hash* funkcija, nakon čega se generiše kod koji se pamti zajedno sa referencom u okviru konkretne grupe podataka (engl. *bucketa)*. Kada se prilikom upita, vrši provera i poklapanje sa nekom vrednošću (WHERE klauzula sa operatorom “=”) korišćenjem iste *hash* funkcije se dolazi do *bucket*-a u okviru kojeg se nalazi podatak, kao i do hash koda i reference. Stoga, ovi indeksi se koriste isključivo prilikom korišćenja operatora jednakosti, jer u suprotnom ne bi bilo moguće primeniti funkciju.

CREATE INDEX *ime\_indeksa* ON *ime\_tabele* USING hash(*ime\_kolone*);

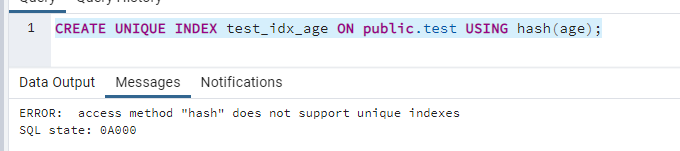
U slučaju kolizije, primenjuje se hash funkcija, kako bi se došlo do celobrojnog koda, gde se nakon toga izvršava funkcija koja vrši raspodelu po *bucket*-ima(*mod(n\_buckets))* kako bi se došlo do grupe u okviru koje se podatak nalazi.

Takođe, moguće je da se te inicijalne grupe definisane hash funkcijom prošire, kada ne bude bilo više prostora, čime dolazi do podele (engl. *split*) indeksa.



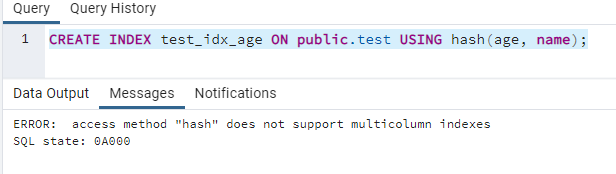
Slika 13. Hash indeksi - podela *bucket*-a

Kod Hash indeksa postoje i određene limitacije. Naime, nije moguće napraviti jedinstvene (engl. Unique) hash indekse.



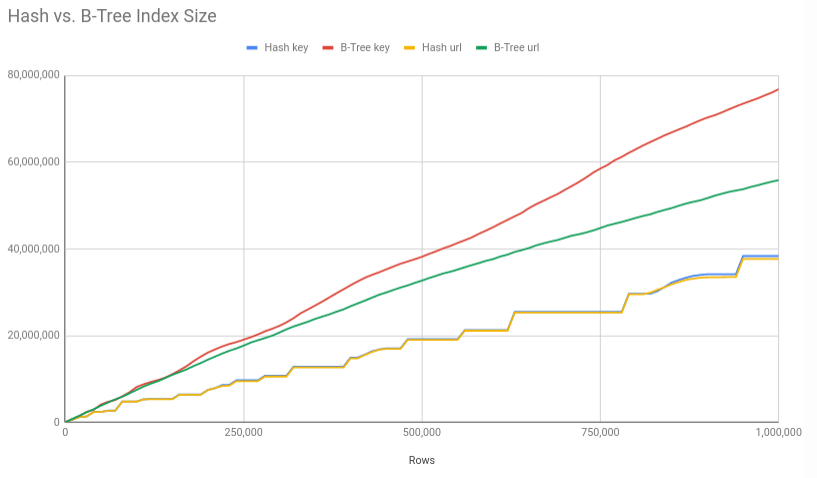
Slika 14. Pokušaj kreiranja jedinstvenog hash indeksa

Nije moguće kreirati ni *multi-column* hash indekse:



Slika 15. *Multi-column* hash indeksi

### 3.3.1 Poređenje B-tree i Hash



Slika 16. Hash i B-tree poređenje

* Hash indeksi zauzimaju manje memorije
* Hash indeksi rastu u inkrementima, dok rast kod B-tree indeksa deluje linearno
* Veličina hash indeksa ne zavisi od vrednosti koja predstavlja vrednost kolone nad kojom je indeks napravljen. Rezultat hash indeksa će biti hash vrednost u svim slučajevima, dok se kod B-tree indeksa konkretna vrednost stavlja u okviru lista stabla
* Generalno, hash indeksi su brži od B-tree indeksa,

Sada ćemo porediti performanse ova dva indeksa prilikom upisa i čitanja. Prvo ćemo napraviti novu tabelu i dodati hash indeks:

**CREATE** **TABLE** shorturl\_hash (

id serial **primary** **key**,

**key** text **not** **null**,

url text **not** **null**

);

**CREATE** **INDEX** shorturl\_hash\_hash\_ix **ON** shorturl\_hash **USING** hash(**key**);

Nakon toga, dodaćemo milion novih redova i izmeriti vreme koje je potrebno za dodavanje tog broja:



Slika 17. Upis milion rekorda sa hash indeksom

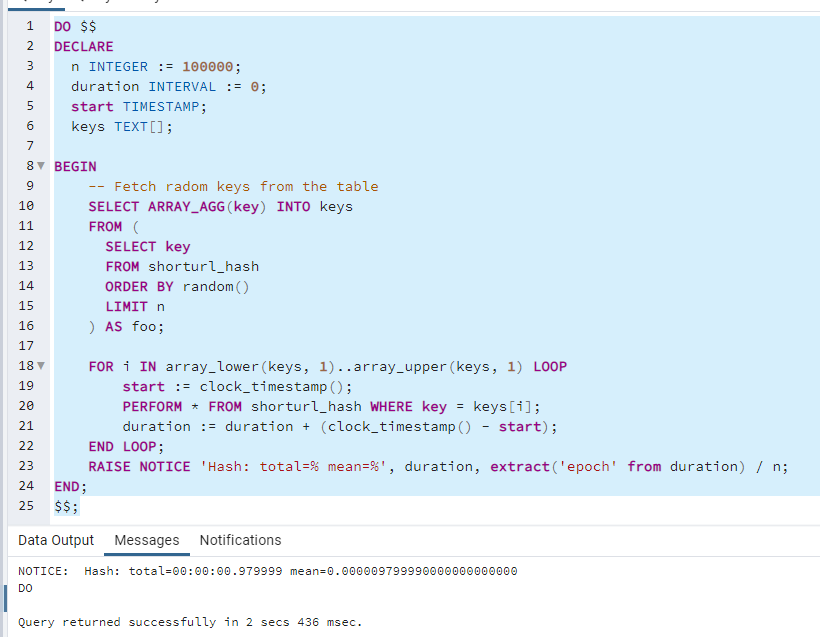
Prema gornjoj slici možemo videti da je upit za upis sa hash indeksom trajao oko 18 sekundi.

Sa druge strane ukoliko napravimo tabelu iste strukture, gde ćemo samo umesto hash indeksa koristiti b-tree indeks:



Slika 18. Upis milion rekorda sa b-tree indeksom

Možemo videti da su rezultati upisa milion rekorda sa B-tree indeksom oko 32 sekunde, tako da možemo zaključiti da je manje vremena potrebno za upis sa hash indeksom.



Slika 19. SELECT 100 000 random redova sa hash indeksom



Slika 20. SELECT 100 000 redova sa B-tree indeksom

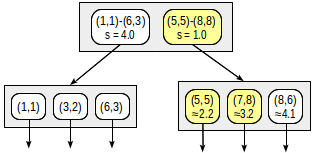
Kao što možemo videti na prethodnim slikama, hash indeks daje bolje rezultate od B-tree indeksa i prilikom čitanja.

## 3.4 GiST

GiST indeks, čije ime predstavlja akronim - *Generalized Search Tree,*  služi kako bi se koristio kod indeksiranja vrednosti koje nije moguće indeksirati uz pomoć B-tree indeksa, koje predstavljaju složene tipove podataka, poput teksta, nizova, dvodimenzionalnih geometrijskih struktura (tačke, poligoni). Ovi indeksi predstavljaju strukturu koja se može koristiti u različitim strategijama, u zavisnosti od implementacije. Po imenu se može zaključiti da se radi o stablu, koje je izbalansirano, koje za razliku od B-tree u čvorovima ne sadrži “obične“, celobrojne vrednosti, već opseg određenih vrednosti, pa iz tog razloga služe za kompleksnije tipove i slučajeve.

Neki od operatora koji se mogu koristiti:

<< &< &> >> <<| &<| |&> |>> @> <@ ~= &&



Slika 21. GiST

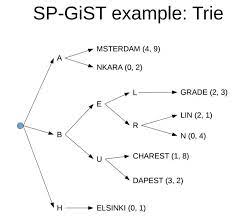
Da bi se kreirao ovaj indeks potrebno je izvršiti komandu:

CREATE INDEX *ime\_indeksa* ON *ime\_tabele* USING gist (*kolona*);

## 3.5 SP-GiST

Slični kao GiST. Služe za pretragu velike količine kompleksnih podataka, kao što su nizovi, tekst, geometrijske strukture. Samo ime predstavlja akronim - *Space-Partitioning Generalized Search Tree*, gde je moguće zaključiti da prilikom njegovog korišćenja dolazi do podele prostora na kome se čuvaju podaci. Koristi se u iste svrhe kao GiST, u okviru nekih specijalizovanih algoritama pretrage, uz korišćenje operatora:

<< >> ~= <@ <<| |>>

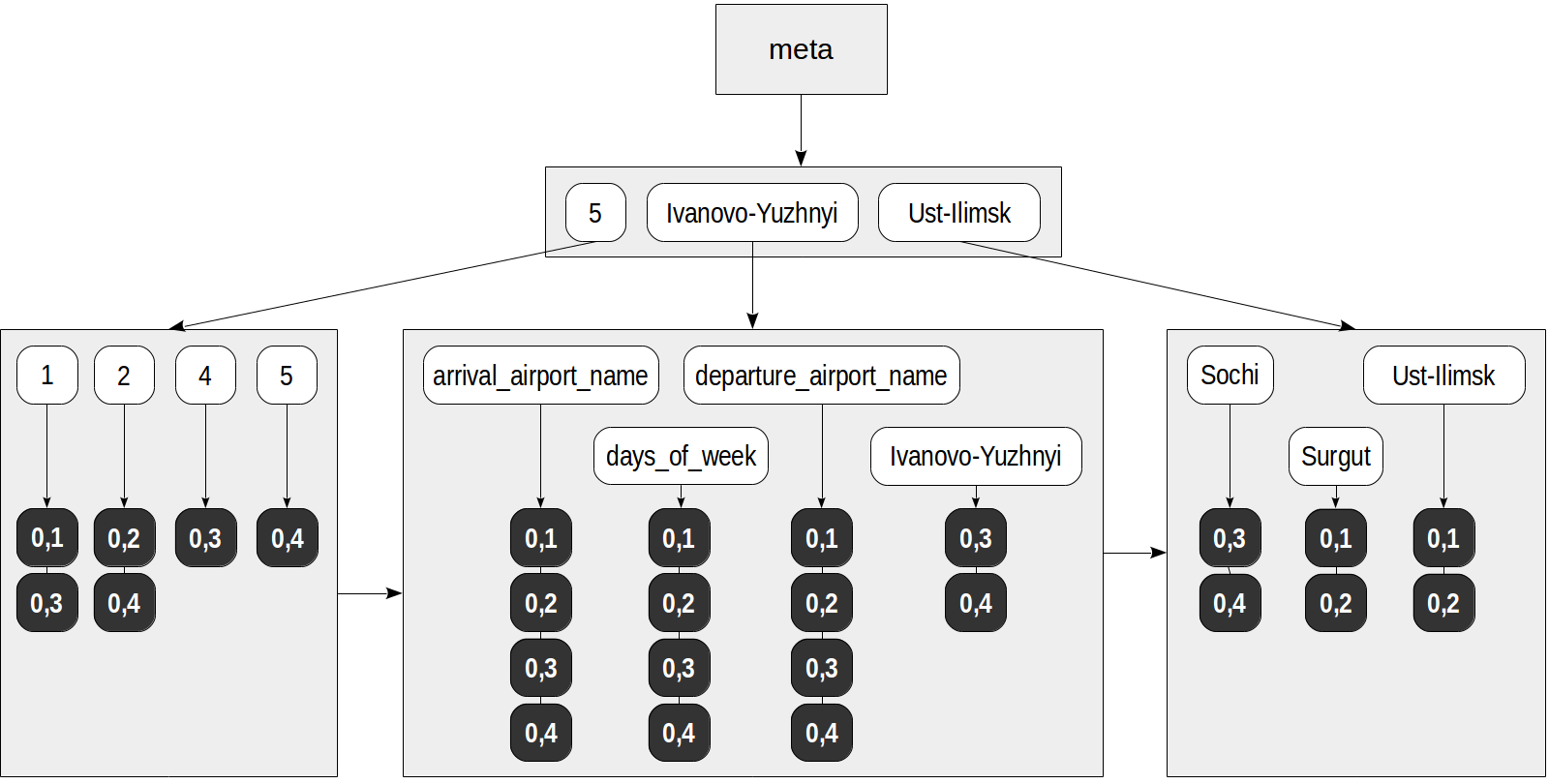


Slika 22. SP-GiST

## 3.6 GIN

GIN indeksi (engl. *Generalized Inverted Index*) predstavljaju takozvane invertovane indekse, koji služe za indeksiranje struktura podataka koje sadrže veći broj vrednosti, poput nizova, JSON objekta . Sadrže parametar za svaku vrednost niza i mogu se koristiti u različitim korisnički definisanim scenarijima. Operatori koji se koriste sa GIN indeksom:

<@ @> = &&



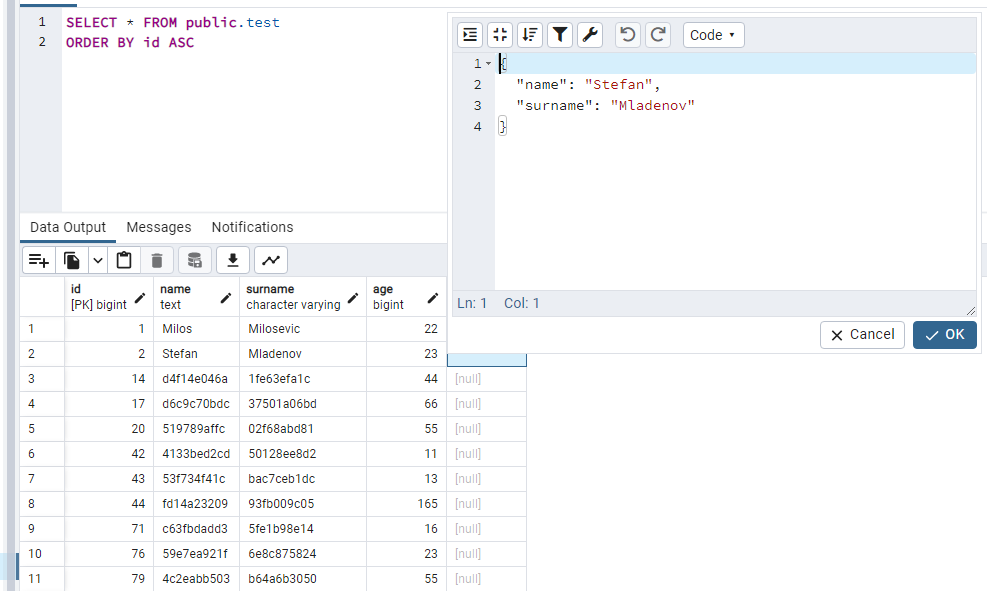
Slika 23. GIN indeks

Da bi se kreirao ovaj indeks potrebno je izvršiti komandu:

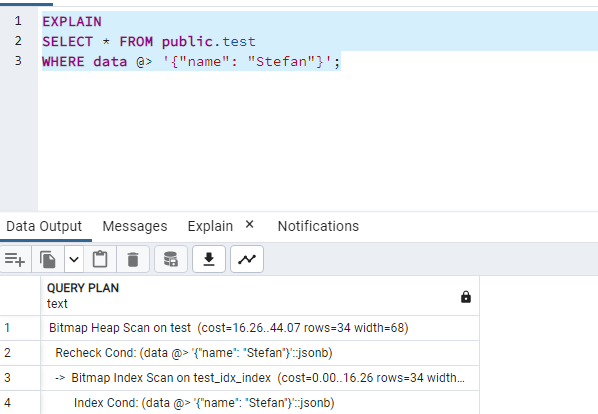
CREATE INDEX *ime\_indeksa* ON *ime\_tabele* USING gin(*kolona*);

Napravićemo indeks nad kolonom pod nazivom *data* tipa *jsonb*:

CREATE INDEX gin\_index ON public.test USING GIN (data);



Slika 24. JSON objekat u tabeli

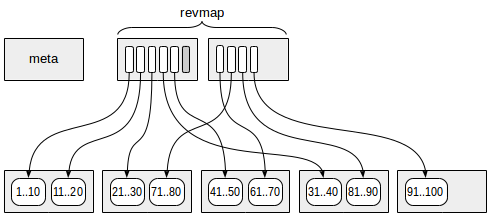


Slika 25. GIN pretraga

Treba biti oprezan prilikom pravljenja ovog tipa indeksa, jer samo kreiranje indeksa nad kolonama sa puno komponenti može dovesti do velikog usporenja upisa i ažuriranja.

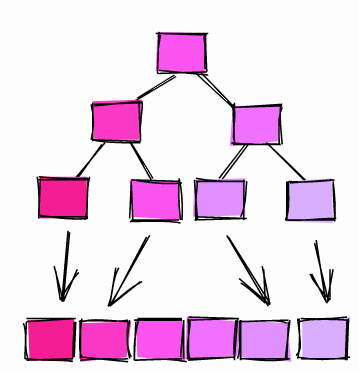
## 3.7 BRIN

BRIN indeksi (engl. *Block Range Index*) su indeksi kod kojih postoji prirodna korelacija sa fizičkom lokacijom, što je recimo slučaj sa ID-jem koji se generiše po automatizmu. Koriste se kako bi se kreirala veza između vrednosti kolone i opsega u okviru kog se vrednost može naći prilikom traženja.

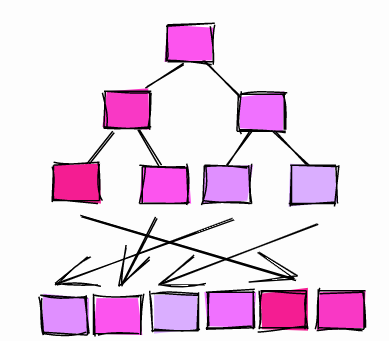


Slika 26. BRIN Indeksi

Korisni su prilikom pretrage velike količine podataka i funkcionišu najbolje kod kolona sa visokom korelacijom. Jedna od statistika koju prikuplja i analizira sama baza podatak jeste korelacija podataka koja predstavlja vezu između redosleda redova u tabeli i njihovih fizičkih lokacija na disku. Može biti u opsegu od -1 do 1, pri čemu važi da što je vrednost korelacije dalja od nule, operacija pretrage po indeksu se smatra jeftinijom. Kada je korelacija vrednosti 1 ili blizu 1, smatra se da su redovi na disku upisani u istom ili sličnom redosledu kao i podaci u samoj tabeli, dok sa druge strane ukoliko je korelacija blizu -1, znači da se podaci na disku nalaze u obrnutom redosledu u odnosu na tabelu. Dobar primer ovakvog slučaja je inkrementalni ID koji se generise po automatizmu ili kolone koje sadrže datum kreiranja.



Slika 27. Korelacija 1



Slika 28. Korelacija 0

BRIN indekse je najbolje koristiti nad kolonama koje su sortirane i klasterovane. Da bi se podaci klasterovali na disku, potrebno je napraviti klasterovani indeks po kome će biti izvršena preraspodela memorijskih lokacija:

CREATE CLUSTERED INDEX *ime\_indeksa* ON *ime\_tabele* (*kolona*);

Ili korišćenjem komande:

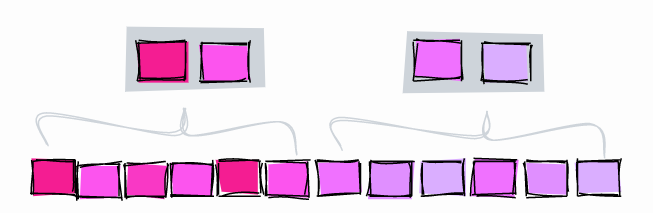
CLUSTER *table\_name*

Pri čemu treba znati da se klasterovanje obavlja u trenutku okidanja komande kod drugog slučaja. Prilikom dodavanja novih i ažuriranja rekorda, baza neće sama izvršiti preraspodelu, već je potrebno ponovo okinuti komandu, što se može desiti i u situaciji sa kreiranjem klasterovanog indeksa.

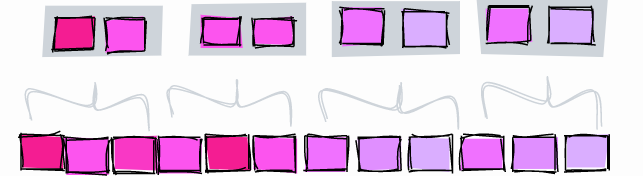
Operatori koji se mogu koristiti sa BRIN indeksom:

< <= = >= >

BRIN indeksi kreiraju opsege, u okviru kojih se vrši pretraga za traženi podatak, gde svaki blok po default-u ima 128 stranica. Što je veći broj stranica po bloku, indeks će biti manji I neefikasniji.



Slika 29. BRIN indeks sa većim brojem stranica po bloku



Slika 30. BRIN indeks sa manjim brojem stranica po bloku

# 4. Zaključak

Indeksiranje baze podataka može biti rešenje ukoliko je potrebno optimizovati i ubrzati upite. Kreiranjem indeksa nad kolonama koje se često javljaju kao uslovi u upitima ili kao delovi spojeva dveju tabela, ubrzava se izvršenje samih upita, pogotovo u slučaju velike količine podataka. Sa druge strane, ovo ne podrazumeva kreiranje indeksa nad svim kolonama, jer održavanje struktura koje se kreiraju u pozadini može biti skupo, tako da je potrebno naći pravi balans. Indeksi ubrzavaju čitanje i pretragu, dok sa druge strane operacije upisa i ažuriranja se mogu izvršavati duži vremenski period. Takođe, brisanje kao i pristupi disku mogu biti duži. PostgreSQL baza pruža širok izbor tipova indeksa, tako da je bitno napraviti dobar izbor prilikom kreiranja samog indeksa za konkretan problem i slučaj korišćenja.

# 5. Reference

[1] <https://www.postgresql.org/docs/>

[2] <https://www.highgo.ca/2020/06/22/types-of-indexes-in-postgresql/>

[3] <https://medium.com/geekculture/indexing-in-postgres-db-4cf502ce1b4e>

[4] <https://www.javatpoint.com/postgresql-indexes>

[5] <https://www.enterprisedb.com/postgres-tutorials/overview-postgresql-indexes>

[6] <https://hakibenita.com/postgresql-hash-index>