PostgreSQL - Interna struktura i organizacija indeksa

-Seminarski rad-

Mentor:
Prof. dr Aleksandar Stanimirović

Student:
Đurađ Gvozdenović 1329

Sadržaj

Uvod

Savremeni dbms sistemi koriste linearnu pretragu za pristupanje zapisima iz baze podataka, ova metoda u proseku ima oslozenost O(n/2) što u zavisnosti količine podataka u bazi može zahtevati značajnu količinu vremena za izvršenje. Glavnu metodu za smanjenje vremena potrebnog da se pristupi zapisu u bazi pretstavljaju indeksi baze podataka.

Indeksi pretstavljaju način da se optimizuju performanse baze podataka minimiziranjem broja pristupa disku koji su potrebni kada se upit obrađuje. To je tehnika zasnovana na strukturama podataka koja se koristi za brzo lociranje i pristup podacima u bazi podataka.

Iako pretstavljaju veoma efikasan mehanizam, treba biti obazriv sa kreiranjem indeksa zbog toga što je prekomernom upotrebom moguće uticati na povećanje vremena koje je potrebno za upis podataka u bazu. Pored toga, kreiranje većeg broja indeksa povećava iskorišćenost diska I glavne memorije što takođe može uticati na smanjenje performansi sistema.

Indeksi

Indeksne strukture su dodatne datoteke na disku koje obezbeđuju sekundarne pristupne putanje, ove pristuone putanje, zatim, omogućavaju alternativne načine pristupa zapisima bez uticaja na fizičku poziciju zapisa u primarnoj datoteci na disku. Oni omogućavaju efikasan pristup zapisima na osnovu indeksnih polja koja se koriste za konstruisanje indeksa. U osnovi, bilo koje polje datoteke se može koristiti za kreiranje indeksa, a više indeksa na različitim poljima, kao i indeksi na više polja, mogu se konstruisati nad istom datotekom. Mogući su različiti indeksi. Svaki od njih koristi određenu strukturu podataka da bi ubrzao pretragu. Da bi se pronašao zapis ili više zapisa u datoteci na osnovu uslova pretrage u polju za indeksiranje, pretražuje se indeks, čime se pronalazi pokazivač na jedan ili više blokova diska u datoteci podataka gde se nalaze potrebni zapisi. Najzastupljeniji tipovi indeksa su zasnovani na uređenim fajlovima (jednonivovski indeksi) i strukturama podataka stabla (višenivovski indeksi, B + -stabla). Indeksi se takođe mogu konstruisati na osnovu heširanja ili drugih struktura podataka pretrage. Takođe postoje I indeksi koji pretstavljaju vektore bitova koji se nazivaju indeksi bitmap-a.

Jednonivovski uređeni indeksi

Ideja koja stoji iza uređenih indeksa je slična onoj iza indeksa koji se koristi u udžbenicima, koji navodi važne termine na kraju knjige po abecednom redu zajedno sa listom brojeva stranica na kojima se termin pojavljuje u knjizi. Možemo da pretražimo indeks knjige za određeni termin u udžbeniku da bismo pronašli listu adresa, u ovom slučaju brojeva stranica, i da koristimo ove adrese da prvo lociramo navedene stranice, a zatim da tražimo termin na svakoj navedenoj stranici. Alternativa, ako se ne daju nikakve druge smernice, bila bi da polako pregledamo ceo udžbenik reč po reč da bismo pronašli termin koji nas zanima; ovo odgovara linearnom pretraživanju, koje skenira celu datoteku. Naravno, većina knjiga ima dodatne informacije, kao što su naslovi poglavlja i odeljaka, koji nam pomažu da pronađemo termin bez potrebe da pretražujemo celu knjigu. Međutim, indeks je jedina tačna indikacija stranica na kojima se svaki termin pojavljuje u knjizi.

Za datoteku sa datom strukturom zapisa koja se sastoji od nekoliko polja (ili atributa), struktura pristupa indeksu se obično definiše nad jednim poljem datoteke, koje se naziva indeksno polje (ili indeksni atribut). Indeks obično skladišti svaku vrednost indeksnog polja zajedno sa listom pokazivača na sve blokove diska koji sadrže zapise sa tom vrednošću polja. Vrednosti u indeksu su poređane tako da možemo da izvršimo binarnu pretragu indeksa. Ako su i datoteka sa podacima i datoteka indeksa sortirani, a pošto je indeksna datoteka obično mnogo manja od datoteke sa podacima, pretraga indeksa pomoću binarne pretrage je bolja opcija.

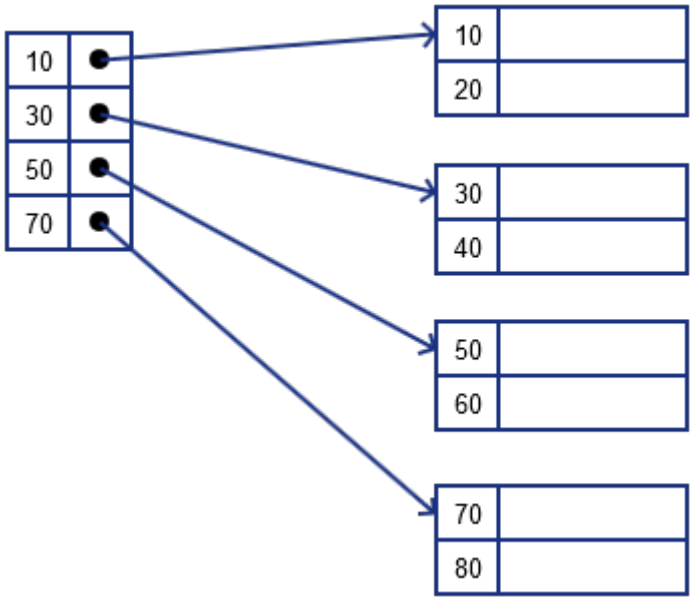
Postoji nekoliko tipova uređenih jednonivovskih indeksa:

1. Primarni indeksi – definisan je nad ključem koji se koristi za sortiranje datoteke sa podacima. Ovaj ključ se koristi kako bi se fizički sortirali podaci na disku, I ima jednistvenu vrednost za svaki zapis.
2. Klastering indeksi – Ovaj tip indeksa koristi se u slučaju da ključ po kome se sortiraju podaci nije jedinstven
3. Sekundarni indeksi – Može biti definisan na bilo kojem ključu koji se ne koristi za fizičko sortiranje zapisa u fajlu. Fajl sa zapisima moye imati više sekundarnih indeksa zajedno sa primarnim ili klastering indeksom.

Treba primetiti da je fajl moguće sortirati samo po jednom ključu, tako da je moguće kreirati ili primarni indeks ili klastering indeks, ali nikako oba zajedno.

Primarni indeksi

Primarni indeks je sortirani fajl čiji su zapisi fiksne dužine koji se sastoje od dva polja, i služi kao pristupna struktura za efikasno traženje i pristup zapisima u fajlu sa podacima. Prvo polje je istog tipa kao polje po kome je sortirana datoteka sa podacima, naziva se primarni ključ, a drugo polje je pokazivač na blok diska (adresa bloka). Postoji jedan indeksni unos (ili indeksni zapis) u fajlu indeksa za svaki blok u fajlu podataka. Svaki indeksni unos ima vrednost polja primarnog ključa za prvi zapis u bloku kao vrednost svog prog polja i pokazivač na taj blok kao vrednost drugog.



Slika 1: Retki indeks

Indeksi se takođe mogu okarakterisati kao gusti ili retki. Gusti indeks ima zapis za svaku vrednost ključa za pretragu (a samim tim i svaki zapis) u fajlu podataka. S druge strane, retki (ili negusti) indeksi imaju zapise samo za neke vrednosti primarnog ključa. Retki indeks ima manje zapisa od broja zapisa u fajlu podataka. Dakle, primarni indeks je negust (retki) indeks, pošto uključuje zapis sa pokazivačem na svaki blok diska fajla sa podacima I vrednošću ključa njegovog vodećeg zapisa, a ne za svaku vrednost primarnog kluča (ili svaki zapis).

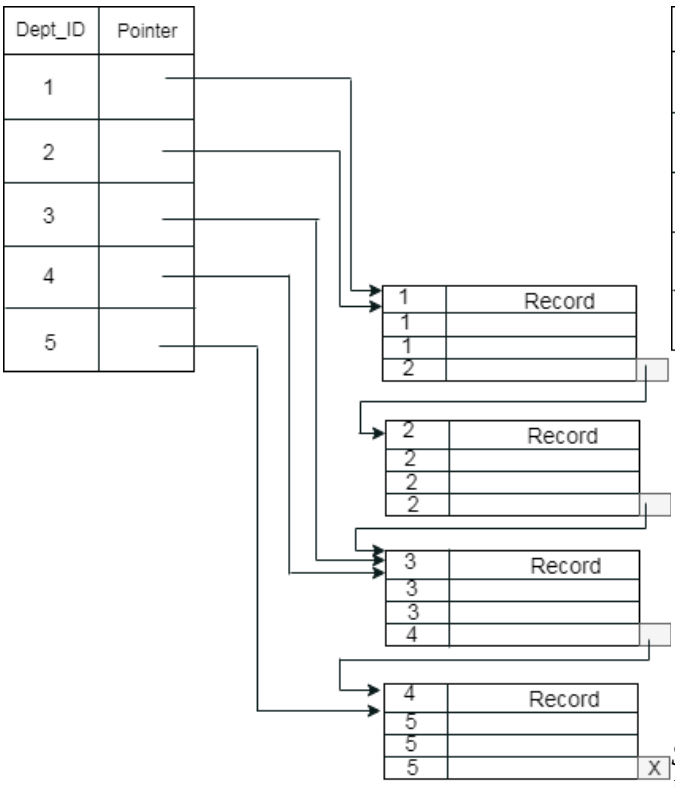
Indeksni fajl za primarni indeks zauzima mnogo manji prostor od fajla sa podacima, iz dva razloga. Prvo, ima manje zapisa u indeksu nego što ima zapisa u fajlu podataka. Drugo, svaki indeksni zapis je obično manji po veličini od zapisa podataka jer ima samo dva polja; shodno tome, više indeksnih zapisa nego zapisa podataka može da stane u jedan blok. Prema tome, binarna pretraga indeksnog fajla zahteva manje pristupa blokovima nego binarna pretraga fajla sa podacima.

Glavni problem sa primarnim indeksom, kao i sa bilo kojim uređenim fajlom, je umetanje i brisanje zapisa. Sa primarnim indeksom, problem se pogoršava jer ako pokušamo da ubacimo zapis na njegovu ispravnu poziciju u fajlu podataka, moramo ne samo da premestimo zapise da bismo napravili prostor za novi zapis, već i da promenimo neke zapise indeksa, pošto će pomeranje zapisa promenite vodeće zapise nekih blokova. Korišćenje neuređenog fajla sa prekoračenjem može smanjiti ovaj problem. Druga mogućnost je da se koristi povezana lista zapisa koji prekoračuju opseg bloka za svaki blok u fajlu podataka. Zapisi unutar svakog bloka i njegova povezana lista mogu se sortirati da bi se poboljšalo vreme pribavljanja. Brisanjem zapisa se upravlja pomoću markera za brisanje.

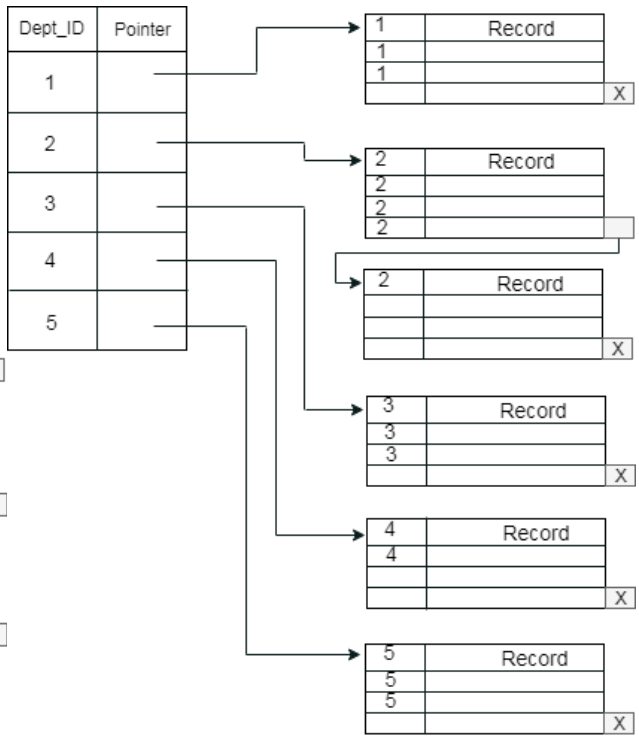
Klastering indeksi

Ako su zapisi u fajlu fizički poređani u odnosu polje koje nije ključ, odnosno koje nema jedinstvenu vrednost za svaki zapis, to polje se naziva polje grupisanja, a fajl podataka se naziva klasterovani fajl. Moguće je kreirati drugačiji tip indeksa, koji se zove klastering indeks, da bi se ubrzalo preuzimanje svih zapisa koji imaju istu vrednost za polje grupisanja. Ovo se razlikuje od primarnog indeksa, koji zahteva da polje po kome se sortira fajl podataka ima jedinstvenu vrednost za svaki zapis.

Klastering indeks je takođe uređen fajl sa dva polja, prvo polje je istog tipa kao polje za grupisanje iz fajla sa podacima, a drugo polje je pokazivač na blok diska. Postoji jedan unos u klastering indeksu za svaku različitu vrednost polja za grupisanje, i on sadrži vrednost i pokazivač na prvi blok u fajlu podataka koji ima zapis sa tom vrednošću za svoje polje grupisanja. Umetanje i brisanje zapisa i dalje izaziva probleme jer su zapisi podataka fizički sortirani. Da bi se ublažio problem umetanja, uobičajeno je da se rezerviše ceo blok (ili klaster susednih blokova) za svaku vrednost polja za grupisanje; svi zapisi sa tom vrednošću se smeštaju u blok (ili blok klaster). Ovo čini umetanje i brisanje relativno jednostavnim.



Slika 2: Klastering indeks



Slika 3: Klastering indeks sa rezervisanim blokovima

Indeks grupisanja je još jedan primer negustog indeksa jer ima zapis za svaku različitu vrednost indeksiranog polja, koje po definiciji nije ključ i stoga mu se vrednosti ponavljaju, odnosno nemaju jedinstvenu vrednost za svaki zapis u datoteci.

Sekundarni indeksi

Sekundarni indeks pruža sekundarni način pristupa fajlu podataka za koju već postoji neki primarni pristup. Zapisi fajla sa podacima mogu biti sortirani, nesortirani ili heširani. Sekundarni indeks može biti kreiran na polju koje je kandidat za ključ i ima jedinstvenu vrednost u svakom zapisu ili

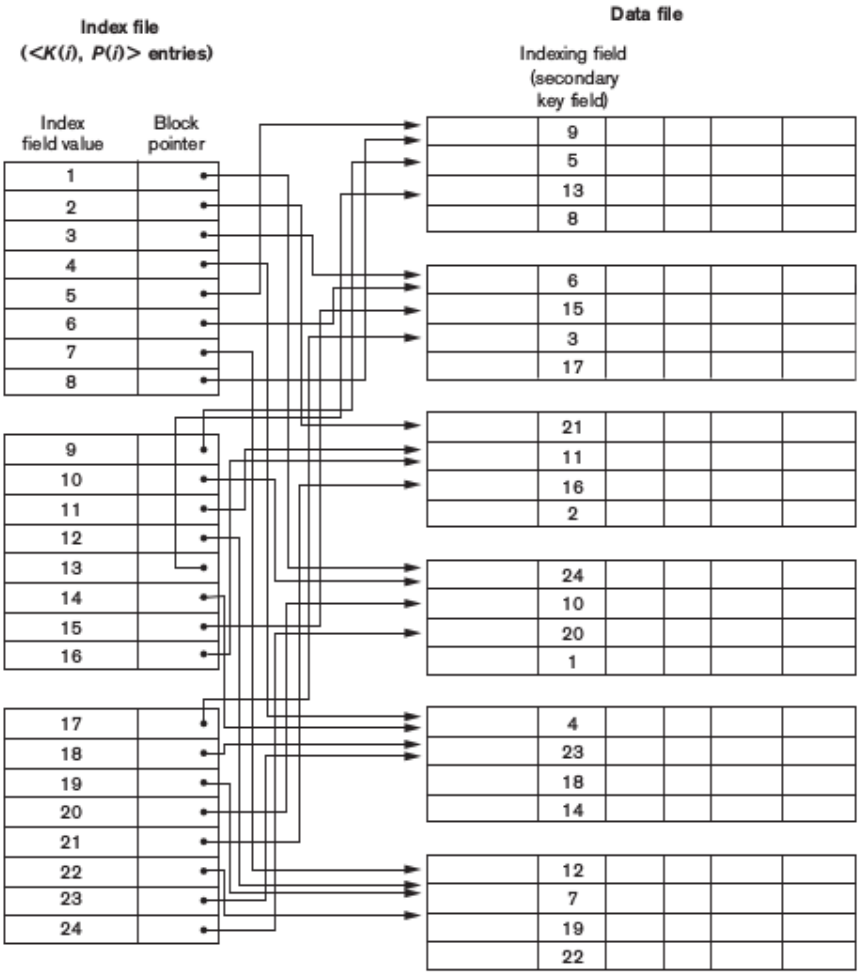
na polju koje nije ključ I sadrži vrednosti koje se ponavljaju. Sekundarni indeks je sortiran fajl sa dva polja. Prvo polje je istog tipa podataka kao neko polje po kome nije uređen fajl podataka I to je polje za indeksiranje. Drugo polje je ili pokazivač bloka ili pokazivač zapisa. Mnogi sekundarni indeksi (a samim tim i polja za indeksiranje) mogu se kreirati za isti fajl, svaki predstavlja dodatni način pristupa toj datoteci na osnovu nekog specifičnog polja.

Prvi tip sekundarnih indeksa su oni koji se fromiraju na ključnom (jedinstvenom) polju koje ima različitu vrednost za svaki zapis. Takvo polje se ponekad naziva sekundarnim ključem; u relacionom modelu, ovo bi odgovaralo bilo kojoj UNIQUE klauzuli ili atributu primarnog ključa tabele. U ovom slučaju postoji jedan indeksni zapis za svaki zapis u fajlu podataka, koji sadrži vrednost polja za zapis i pokazivač ili na blok u kome je zapis uskladišten ili na sam zapis. Dakle, takav indeks je gust.

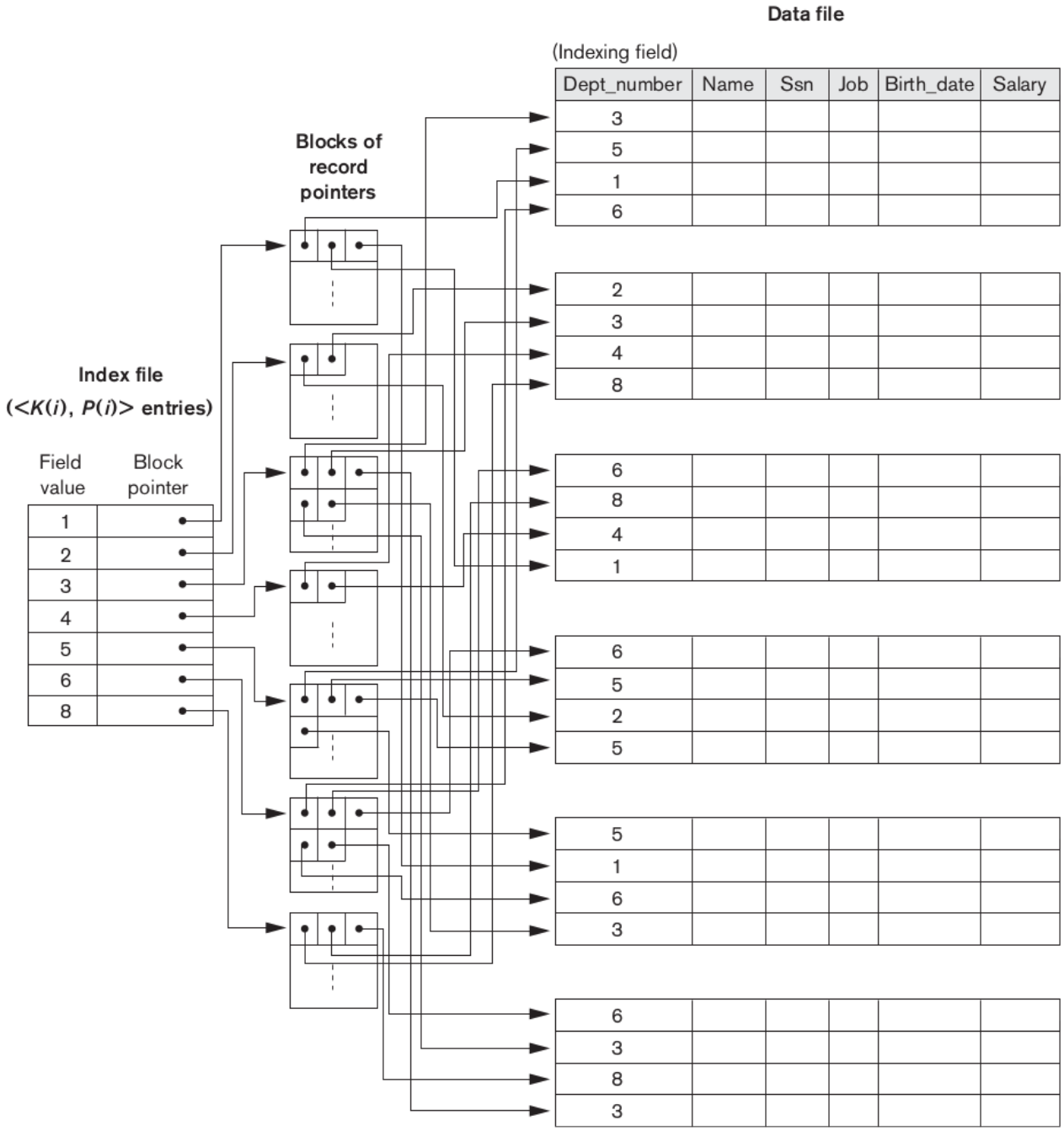
Zapisi indeksa su sortirani po vrednosti indeksiranog polja, tako da je moguće vršiti binarnu pretragu. Pošto zapisi fajla sa podacima nisu fizički poređani prema vrednostima polja sekundarnog ključa, ne možemo koristiti tehniku vodećeg zapisa iz bloka. Zbog toga se zapis indeksa kreira za svaki zapis u fajlu podataka, a ne za svaki blok, kao u slučaju primarnog indeksa. Kada se odgovarajući blok diska prenese u glavni memorijski bafer, može se izvršiti traženje željenog zapisa unutar bloka.

Sekundarni indeks obično zahteva više prostora za skladištenje i duže vreme pretrage nego primarni indeks, zbog većeg broja zapisa. Međutim, poboljšanje vremena pretrage za proizvoljni zapis je mnogo veće za sekundarni indeks nego za primarni indeks, zbog toga što bi bilo nephodno izvršiti linearnu pretragu fajla sa podacima da sekundarni indeks ne postoji. Za primarni indeks, i dalje bi bilo moguće koristiti binarnu pretragu po glavnoj datoteci, čak i ako indeks ne postoji.

Sekundarni indeks obezbeđuje logički redosled zapisa prema polju za indeksiranje. Ako se pristupa zapisima po redosledu unosa u sekundarnom indeksu, oni će se dobiti po redosledu polja za indeksiranje. Primarni indeksi i klastering indeksi pretpostavljaju da je polje koje se koristi za fizičko sortiranje zapisa u fajlu isto kao i polje za indeksiranje.



Slika 4: Sekundarni indeks nad poljem sa jedinstvenim vrednostima



Slika 5: Sekundarni indeks nad poljem čije vrednosti nisu jedinstvene

Višenivovski indeksi

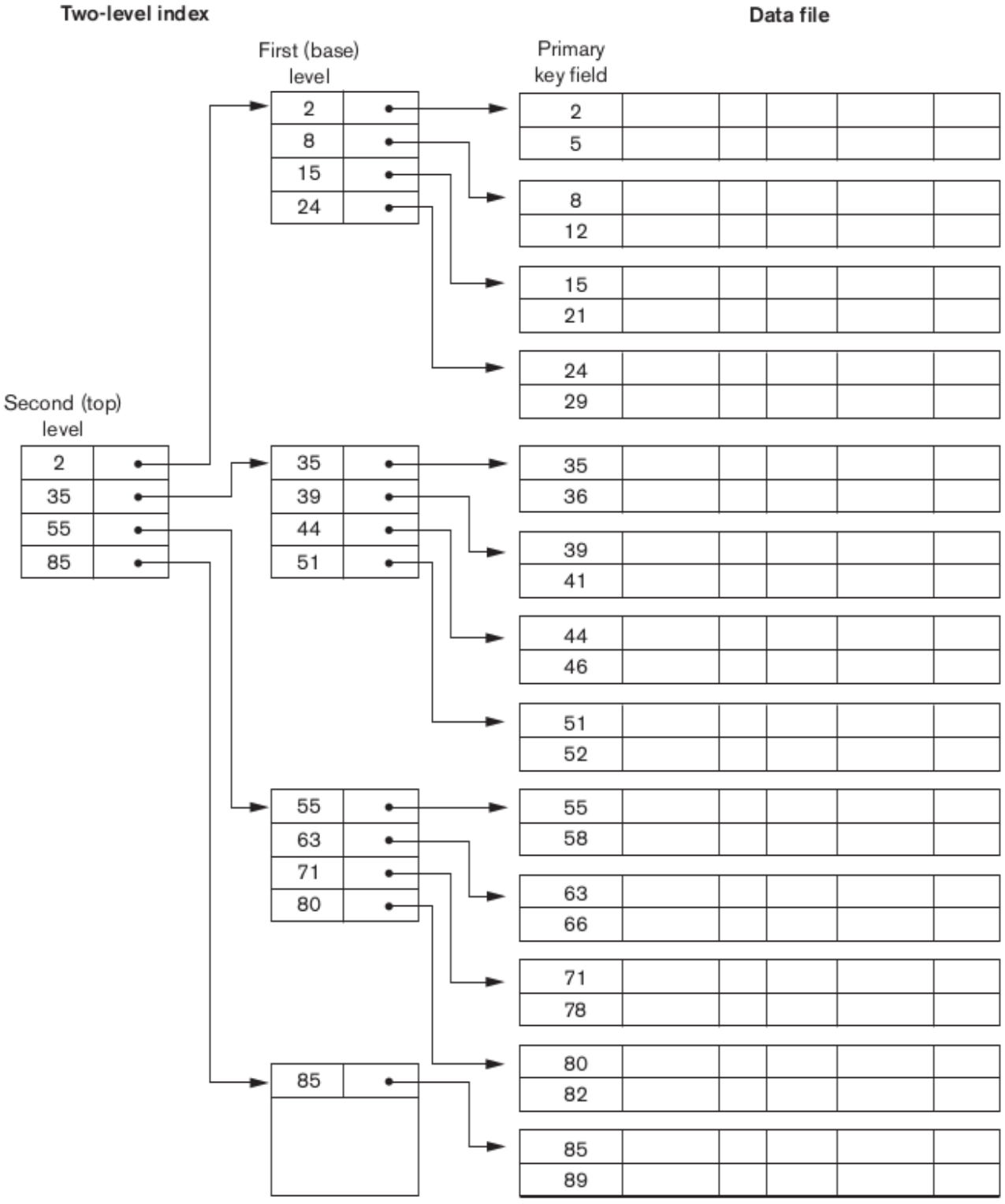
Šeme indeksiranja kod jednonivovskih indeksa uključuju uređenu indeksnu datoteku. Binarno pretraživanje se primenjuje nad indeksom da bi se locirali pokazivači na blok diska ili na zapis (ili na više zapisa) u datoteci koji ima određenu vrednost indeksnog polja. Binarno pretraživanje zahteva približno ($\log_2 b_i$) pristupa bloku za indeks sa b_i blokova jer svaki korak algoritma smanjuje deo indeksnog fajla koji se koristiu nastavku pretraživanja za faktor 2. Zbog toga se uzima funkcija logaritma sa osnovom 2. Ideja iza indeksa na više nivoa je da se smanji deo indeksa koji se koristi u nastavku pretraživanja pomoću bfr_i, faktora blokiranja za indeks, koji je veći od 2. Dakle, prostor za pretragu se smanjuje mnogo brže. Vrednost bfr_i se naziva “fan-out” višenivovskog indeksa, ili skraćeno simbolom fo. Dok se prostor za pretragu zapisa deli na dve polovine u svakom koraku

tokom binarne pretrage, kod višenivovskih indeksa se deli na n delova (gde je n = fan-out) u svakom koraku pretrage. Pretraživanje višenivovskog indeksa zahteva približno ($\log_{fo} b_i$) pristupa bloku, što je znatno manji broj nego za binarnu pretragu ako je fan-out veći od 2. U većini slučajeva, fan-out je mnogo veći od 2.

Višenivovski indeks smatra indeksni fajl (ili prvi, dnosno, osnovni nivo višenivovskog indeksa) kao uređen fajl sa različitom vrednošću za svaki kluč. Stoga, ukoliko se indeksni fajl prvog nivoa posmatra kao sorirani fajl sa podacima, moguće je kreirati primarni indeks za prvi nivo; ovaj indeks naziva se drugi nivo višenivovskog indeksa. Pošto je drugi nivo primarni indeks, možemo koristiti vodeći zapis bloka tako da drugi nivo ima jedan unos za svaki blok prvog nivoa. Blokirajući faktor bfr_i za drugi nivo, kao i za sve naredne nivoe, je isti kao onaj za indeks prvog nivoa jer su svi unosi indeksa iste veličine; svaki ima jednu vrednost polja i jednu adresu bloka. Ako prvi nivo ima r_1 zapisa, a faktor blokiranja, koji je ujedno i fan-out, za indeks je bfr_i = fo, onda je za prvi nivo potrebno $\lceil (r_1 /fo) \rceil$ blokova, što je prema tome broj zapisa r_2 potrebnih na drugom nivou indeksa.

Moguće je ponoviti ovaj proces za drugi nivo. Treći nivo, koji je primarni indeks za drugi nivo, ima zapis za svaki blok drugog nivoa, tako da je broj zapisa trećeg nivoa $r_3 = \lceil (r_2 /fo) \rceil$. Treba primetiti da je potreban drugi nivo samo ako je za prvi nivo potrebno više od jednog bloka za skladištenje na disku, i, shodno tome, potreban je treći nivo samo ako je drugom nivou potrebno više od jednog bloka. Moguće je ponavljati prethodni proces sve dok svi unosi nekog nivoa indeksa t ne stanu u jedan blok. Ovaj blok na t-tom nivou naziva se najviši nivo indeksa. Svaki nivo smanjuje broj zapisa na prethodnom nivou za faktor fo, indeks fan-out, tako da je moguće koristiti formulu $1 \leq (r_1 /((fo)^t))$ za izračunavanje t. Dakle, indeks na više nivoa sa r_1 unosa prvog nivoa će imati približno t nivoa, gde je $t = \lceil (\log_{fo} (r_1)) \rceil$. Prilikom pretraživanja indeksa, na svakom nivou se preuzima jedan blok diska. Dakle, t blokovima diska se pristupa za indeksnu pretragu, gde je t broj nivoa indeksa.

Višenivovska šema koja je ovde opisana može se koristiti za bilo koji tip indeksa, bilo da je primarni, klastering ili sekundarni, sve dok indeks prvog nivoa ima različite vrednosti za kluč i zapise fiksne dužine.



Slika 6: Višenivovski indeks

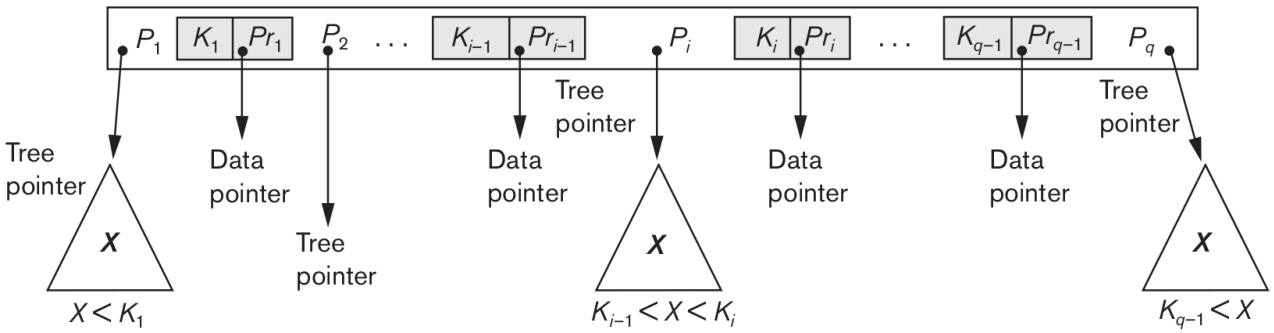
B stabla

B-stablo je stablo pretrage koje je uvek balansirano i za koje važi da prostor izgubljen brisanjem, ako ga ima, nikada ne postane preveliki. Algoritmi za umetanje i brisanje, međutim, postaju složeniji kako bi se održala ova ograničenja. Ipak, većina umetanja i brisanja su jednostavni procesi; oni postaju komplikovani samo pod posebnim okolnostima, naime, kad god se pokuša umetanje u čvor koji je već pun ili bisanje iz čvora koji ga čini manje od pola punim.

B-stablo počinje sa jednim korenskim čvorom (koji je takođe listni čvor) na nivou 0. Kada se osnovni čvor napuni sa p – 1 vrednosti ključa za pretragu i pokuša se ubacivanje još jedanog unosa u stablo, korenski čvor se deli na dva čvora na nivou 1. Samo srednja vrednost se čuva u korenskom čvoru, a ostatak vrednosti su ravnomerno podeljene između druga dva čvora. Kada je ne-korenski čvor pun i u njega se ubaci novi unos, taj čvor se deli na dva čvora na istom nivou, a srednji unos se pomera na roditeljski čvor zajedno sa dva pokazivača na nove podeljene čvorove. Ako je roditeljski čvor pun, on se takođe deli. Razdvajanje se može širiti sve do korenskog čvora, stvarajući novi nivo ako je i koren podeli.

Ako brisanje vrednosti prouzrokuje da je čvor manje od polovine pun, on se kombinuje sa susednim čvorovima, a to se takođe može propagirati sve do korena. Dakle, brisanje može smanjiti broj nivoa stabla. Analizom i simulacijom je pokazano da su, nakon brojnih nasumičnih umetanja i brisanja na B-stablu, čvorovi puni oko 69 procenata kada se broj vrednosti u stablu stabilizuje. Ovo važi i za B+ stabla. Ako se to dogodi, razdvajanje i kombinovanje čvorova će se dešavati samo retko, tako da umetanje i brisanje postaju prilično efikasni. Ako broj vrednosti raste, stablo će se proširiti bez problema, iako može doći do deljanja čvorova, tako da će za neka umetanja biti potrebno više vremena. Svaki čvor B-stabla može imati najviše p pokazivača na podstabla, p – 1 pokazivača na podatke i p – 1 vrednosti polja ključa za pretragu.

Generalno, čvor B-stabla može da sadrži dodatne informacije potrebne algoritmima koji manipulišu stablom, kao što je broj zapisa k u čvoru i pokazivač na roditeljski čvor.



Slika 7: Struktura B stabla

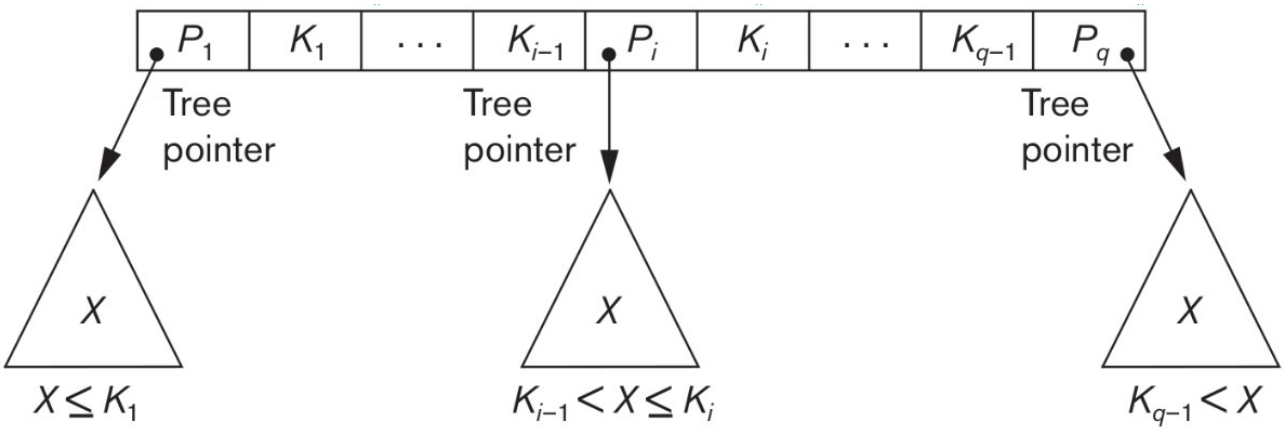
B+ stabla

Većina implementacija višenivoskih indeksa koristi varijaciju strukture podataka B-stabla koja se naziva B+ stablo. U B-stablu, svaka vrednost polja za pretragu se pojavljuje jednom na nekom nivou u stablu, zajedno sa pokazivačem podataka. U B+ stablu, pokazivači podataka se čuvaju samo na lisnim čvorovima stabla; stoga se struktura lisnih čvorova razlikuje od strukture unutrašnjih čvorova. Listovi imaju zapis za svaku vrednost polja za pretragu, zajedno sa pokazivačem podataka na zapis (ili na blok koji sadrži ovaj zapis) ako je polje za pretragu ključno polje. Za polje za pretragu bez ključa, pokazivač pokazuje na blok koji sadrži pokazivače na zapise fajla sa podacima, stvarajući dodatni nivo indirektnosti.

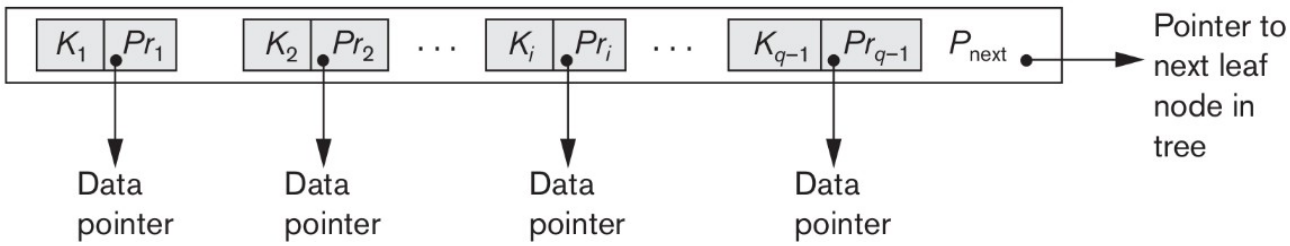
Listni čvorovi B+ stabla su obično povezani da bi se u polju za pretragu obezbedio uređeni pristup zapisima. Ovi lisni čvorovi su slični prvom nivou indeksa. Unutrašnji čvorovi B+ stabla odgovaraju drugim nivoima višenivovskog indeksa. Neke vrednosti polja za pretragu iz listovnih čvorova se ponavljaju u unutrašnjim čvorovima B+ stabla da bi se vodila pretraga.

Pokazivači u unutrašnjim čvorovima su pokazivači stabla koji pokazuju na blokove koji su čvorovi stabla, dok su pokazivači u lisnim čvorovima pokazivači podataka na zapise ili blokove fajla sa podacima, osim P_{next} pokazivača, koji je pokazivač stabla na sledeći lisni čvor. Počevši od krajnjeg levog lisnog čvora, moguće je preći listove čvorova kao povezanu listu, koristeći pokazivače P_{next}. Ovo obezbeđuje uređen pristup zapisima podataka u polju za indeksiranje. $P_{previous}$ pokazivač takođe može biti uključen. Za B+ stablo na polju bez ključa, potreban je dodatni nivo indirektnosti, tako da su P_r pokazivači pokazivači bloka na blokove koji sadrže skup pokazivača zapisa na stvarne zapise u fajlu podataka.

Pošto zapisi u unutrašnjim čvorovima B+ stabla uključuju vrednosti pretrage i pokazivače stabla bez ikakvih pokazivača podataka, više unosa se može spakovati u unutrašnji čvor B+ stabla nego za slično B stablo. Dakle, za istu veličinu bloka, p će biti veće za B+ stablo nego za B stablo. Ovo može dovesti do manjeg nivoa B+ stabla, smanjujući vreme za pretragu. Pošto su strukture za unutrašnje i lisne čvorove kod B+ stabla različite, p može biti različito.



Slika 8: Struktura internih čvorova B+ stabla



Slika 9: Struktura listova B+ stabla

Varijacije B I B+ stabala

U nekim slučajevima, ograničenje 5 na B stablo (ili za unutrašnje čvorove B+ stabla, osim korenskog čvora), koje zahteva da svaki čvor bude bar do pola pun, može se promeniti tako da zahteva da svaki čvor bude najmanje dve trećine pun. U ovom slučaju, B stablo se naziva B* stablo. Generalno, neki sistemi dozvoljavaju korisniku da izabere faktor popunjavanja između 0,5 i 1,0, gde 1,0 znači da čvorovi B stabla moraju biti potpuno puni. Takođe je moguće navesti dva faktora popunjavanja za B+ stablo: jedan za nivo lista i jedan za unutrašnje čvorove stabla. Kada se indeks prvi put konstruiše, svaki čvor se popunjava do približno navedenih faktora popunjavanja. Neki

istraživači su predložili da se olabavi zahtev da je čvor napola pun, i umesto toga da se dozvoli da čvor postane potpuno prazan pre spajanja, kako bi se pojednostavio algoritam za brisanje. Studije I simulacije pokazuju da se time ne gubi previše dodatnog prostora kod nasumičnog umetanja i brisanja.

Metode višestrukog pristupa ključevima

U prethodnoj diskusiji, postoji pretpostavka da su primarni ili sekundarni ključevi po kojima se pristupa fajlovima pojedinačni atributi. U mnogim zahtevima za preuzimanje i ažuriranje uključeno je više atributa. Ako se određena kombinacija atributa često koristi, korisno je postaviti strukturu pristupa kako bi se obezbedio efikasan pristup ključnoj vrednosti koja je kombinacija tih atributa.

Uređeni indeks na većem broju atributa

Moguće je kreirati indeks na polju ključa za pretragu koje se sastoji od kombinacija bilo kojih atributa. Uopšteno govoreći, ako je indeks kreiran na atributima $<A_1 , A_2 , ..., A_n >$, vrednosti ključa za pretragu su torke sa n vrednosti: $<v_1 , v_2 , ..., v_n >$.

Leksikografski poredak ovih vrednosti torke uspostavlja red na ovom kompozitnom ključu za pretragu. Leksikografsko poređanje funkcioniše slično kao i redosled nizova znakova. Indeks na kompozitnom ključu od n atributa funkcioniše slično kao i svaki indeks o kome se do sada govorilo.

Particionisanno heširanje

Particionirano heširanje je proširenje statičkog eksternog heširanja koje omogućava pristup većem broju ključeva. Pogodan je samo za poređenja jednakosti; upiti opsega nisu podržani. U particionisanom heširanju, za ključ koji se sastoji od n komponenti, heš funkcija je dizajnirana da proizvede rezultat sa n odvojenih heš adresa. Adresa celog skupa je konkatenacija ovih n adresa. Tada je moguće tražiti traženi kompozitni ključ za pretragu tako što će se potražiti odgovarajući segmenti koji odgovaraju odgovarajućim delovima adrese.

Grid fajlovi

Druga alternativa je da se fajl podataka organizuje kao grid fajl. Ukoliko jepotrebno pristupiti fajlu uz pomoć dva ključa, moguće je da konstruisati niz mreže sa jednom linearnom skalom (ili dimenzijom) za svaki od atributa pretrage. Sklale su napravljene tako da se postigne uravnotežena raspodela tog atributa. Svaka ćelija ukazuje na neku adresu grupe gde se čuvaju zapisi koji odgovaraju toj ćeliji.

Dugi tipovi indeksa

Haš indeksi

Takođe je moguće kreirati strukture pristupa slične indeksima koje se zasnivaju na heširanju. Heš indeks je sekundarna struktura za pristup fajlu korišćenjem heširanja na ključu za pretragu koji nije

onaj koji se koristi za organizaciju primarnog fajla podataka. Zapisi indeksa su tipa <K, P_r> ili <K, P>, gde je P_r pokazivač na zapis koji sadrži ključ, a P je pokazivač na blok koji sadrži zapis za taj ključ. Indeksni fajl sa ovim indeksnim zapisima može biti organizovan kao dinamički proširiv heš fajl, Traženje zapisa koristi algoritam za pretragu heš-a na K. Kada se unos pronađe, pokazivač P_r (ili P) se koristi za lociranje odgovarajućeg zapisa u fajlu podataka.

Indeksi bitmapa

Indeks bitmapa je još jedna popularna struktura podataka koja olakšava upite po većem broju ključeva. Bitmap indeksiranje se koristi za relacije koje sadrže veliki broj redova. Ovom tehnikom se kreira indeks za jednu ili više kolona, a svaka vrednost ili opseg vrednosti u tim kolonama se indeksira. Tipično, bitmap indeks se kreira za one kolone koje sadrže prilično mali broj jedinstvenih vrednosti. Da bi se napravo indeks bitmape na skupu zapisa u relaciji, zapisi moraju biti numerisani od 0 do n sa ID-om (ID zapisa ili ID reda) koji se može mapirati na fizičku adresu napravljenu od broja bloka i pomerajem zapisa unutar bloka. Indeks bitmapa je izgrađen na jednoj određenoj vrednosti određenog polja (kolona u relaciji) i pretstavjla niz bitova. Uzevši u obzir bitmap indeks za kolonu C i vrednost V za tu kolonu. Za relaciju sa n redova, koja sadrži n bitova. I-ti bit je postavljen na 1 ako red i ima vrednost V za kolonu C; u suprotnom, postavlja se na 0. Ako C sadrži skup vrednosti <v_1, v_2, ..., v_m > sa m različitih vrednosti, tada bi za tu kolonu bilo kreirano m bitmap indeksa.

Indeksi zasnovani na funkcijama

Ideja iza indeksiranja zasnovanog na funkcijama je kreiranje indeksa tako da vrednost koja je rezultat primene neke funkcije nad poljem ili kolekcijom polja postane ključ za indeks.

Indeksi u PostgreSQL-u

PostgreSQL pruža nekoliko tipova indeksa: B stabla, Hash, GiST, SP-GiST, GIN i BRIN. Svaki tip indeksa koristi drugačiji algoritam koji je najprikladniji za različite tipove upita. Podrazumevano, komanda CREATE INDEX kreira indeks tipa B stabla, koji odgovaraju najčešćim situacijama.

B-Tree

PostgreSQL uključuje implementaciju standardne strukture indeksnih podataka btree (višesmerno izbalansirano stablo). Bilo koji tip podataka koji se može sortirati u dobro definisani linearni poredak može biti indeksiran indeksom btree. Jedino ograničenje je da zapis u indeksu ne može da pređe približno jednu trećinu stranice

PostgreSQL B-Tree indeksi su strukture stabla na više nivoa, gde svaki nivo stabla može da se koristi kao dvostruko povezana lista stranica. Jedna metastranica se čuva na fiksnoj poziciji na početku datoteke prvog segmenta indeksa. Sve ostale stranice su ili listovi ili interne stranice. Listne stranice su stranice na najnižem nivou stabla. Svi ostali nivoi se sastoje od internih stranica. Svaka stranica lista sadrži torke koji ukazuju na redove tabele. Svaka interna stranica sadrži tuple koji ukazuju na sledeći nivo niže u stablu. Tipično, preko 99% svih stranica su listovi.

S obzirom na to da se svi pokazivači na podatke opisani gore nalaze u lisnim čvorovima stabla može se zaključiti da implementacija PostgreSQL btree zapravo pretstavlja B+ stablo.

Primer

Za potrebe demostracije rada B-Tree indeksa iskorišćena je tabela scalars u koju je upisano 100 000 skalarnih vrednosti sa nasumičnim vrednostima za kolonu x:

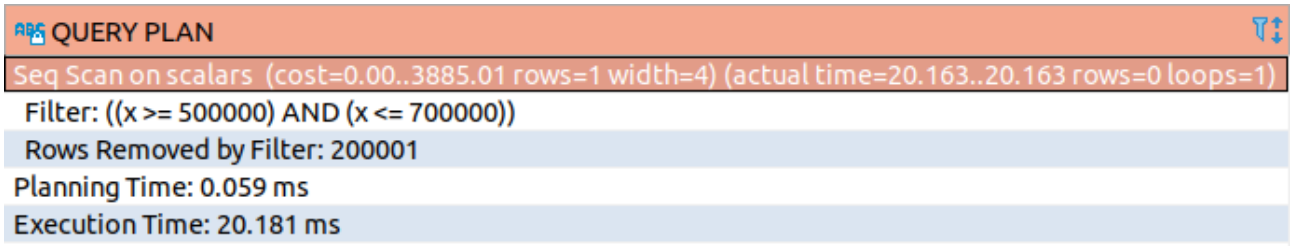
```
conn = psycopg2.connect(
    host="localhost",
    database="dbms",
    user=[redacted]
    password=[redacted]

mycursor = conn.cursor()
for i in range(10**5):
    x = random.randrange(0, 10**5)
    sql = f"INSERT INTO scalars (x) VALUES ({x})"
    mycursor.execute(sql)
conn.commit()
```

Slika 10: Primer koda

Nakon toga izvršen je sledeći upit:

```
explain analyze select * from scalars where x between 500000 and 700000;
```

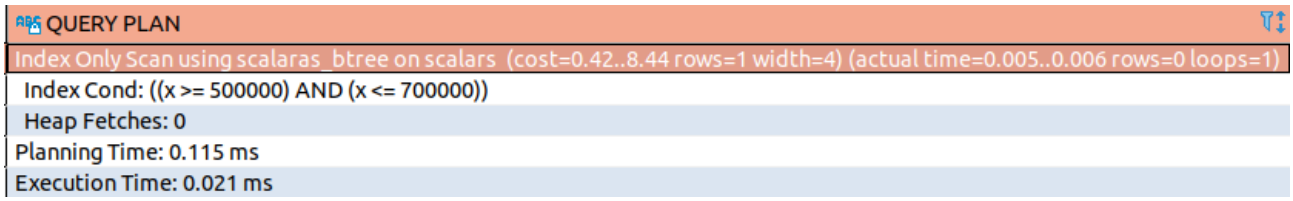
Rezultat izvršenja prikazan je na sledećoj slici:



Slika 11: Rezultat izvršenja

Nakon toga, naredbom `create index scalaras_btree on scalars(x);` kreiran je B-Tree indeks.

Nakon kreiranog indeksa ukoliko se gornji upit ponovo izvrši dobija se sledeći rezultat:



Slika 12: Rezultat izvršenja

Na osnovu datog primera može se zaključiti da je korišćenje B-Tree indeksa u ovom slučaju potpuno opravdano s obzirom na to da je vreme izvršenja upita skraćeno približno 10 puta.

GiST

GiST je skraćenica za Generalizovano stablo pretrage(Generalized Search Tree). To je izbalansiran metod pristupa sa strukturom stabla, koji deluje kao osnovni šablon u kojem se implementiraju proizvoljne šeme indeksiranja. B-stabla, R-stabla i mnoge druge šeme indeksiranja mogu se implementirati u GiST-u. Jedna od prednosti GiST-a je što omogućava razvoj prilagođenih tipova podataka sa odgovarajućim metodama pristupa.

Tradicionalno, implementacija novog metoda pristupa indeksu značila je mnogo teškog posla. Bilo je neophodno razumeti unutrašnje funkcionisanje baze podataka, kao što su menadžer zaključavanja i evidencija unapred. GiST interfejs ima visok nivo apstrakcije, zahtevajući od programera metoda pristupa samo da primeni semantiku tipa podataka kome se pristupa. GiST sloj se sam brine o konkurentnosti, evidentiranju i traženju strukture stabla.

Ovu proširivost ne treba mešati sa proširivosti drugih standardnih stabala pretrage u smislu podataka sa kojima mogu da rukuju. Na primer, PostgreSQL podržava proširiva B-stabla i heš indekse. To znači da je moguće koristiti PostgreSQL da bi se napravilo B-stablo ili heš nad bilo kojim tipom podataka. Ali B-stabla podržavaju samo predikate opsega (<, =, >), a heš indeksi podržavaju samo upite jednakosti.

Dakle, ako se indeksira, recimo, kolekciju slika sa PostgreSQL B-stablom, moguće je postavljati samo upite kao što su „da li je slika X jednaka slici Y“, „da li je slika X manja od slike Y“ i „da li je slika X veća nego slika Y”. U zavisnosti od toga kako definišete „jednako“, „manje od“ i „veće od“ u ovom kontekstu, ovo bi moglo biti korisno. Međutim, korišćenjem indeksa zasnovanog na GiST-

u, bilo bi moguće kreirati načine za postavljanje pitanja specifičnih za domen, kao što su, recimo, “sve slike konja“ ili „pronađi sve slike sa prevelikom ekspozicijom“.

Sve što je potrebno da bi se GiST pristupni metod pokrenuo je implementacija nekoliko korisnički definisanih metoda, koje definišu ponašanje ključeva u stablu. Naravno, ove metode moraju biti prilično dobre da bi podržale komplikovane upite, ali za sve standardne upite (B-stabla, R-stabla, itd.) su relativno jednostavne. Ukratko, GiST kombinuje proširivost zajedno sa opštošću, ponovnom upotrebom koda i čistim interfejsom.

Primer

Za potrebe demostracije rada GiST indeksa iskorišćena je tabela points u koju je upisano 100 000 tačaka sa nasumičnim vrednostima za x i y koordinate:

```
conn = psycopg2.connect(
    host="localhost",
    database="dbms",
    user=[redacted]
    password=[redacted]

mycursor = conn.cursor()
for i in range(10**5):
    x,y = random.randrange(0, 100),random.randrange(0, 100)
    sql = "INSERT INTO points (p) VALUES (point (%s, %s))"
    val = (x, y)
    mycursor.execute(sql, val)
conn.commit()
```

Slika 13: Primer koda

Nakon toga izvršen je sledeći upit:

```
explain analyze select * from points where p <@ box '(2,1),(7,4)';
```

Rezultat izvršenja prikazan je na sledećoj slici:

QUERY PLAN
Gather (cost=1000.00..11326.00 rows=964 width=16) (actual time=0.325..32.181 rows=2361 loops=1)
Workers Planned: 2
Workers Launched: 2
-> Parallel Seq Scan on points (cost=0.00..10229.60 rows=402 width=16) (actual time=0.040..26.866 rows=787 loops=3)
Filter: (p <@ '(7,4),(2,1)'::box)
Rows Removed by Filter: 320468
Planning Time: 0.059 ms
Execution Time: 32.297 ms

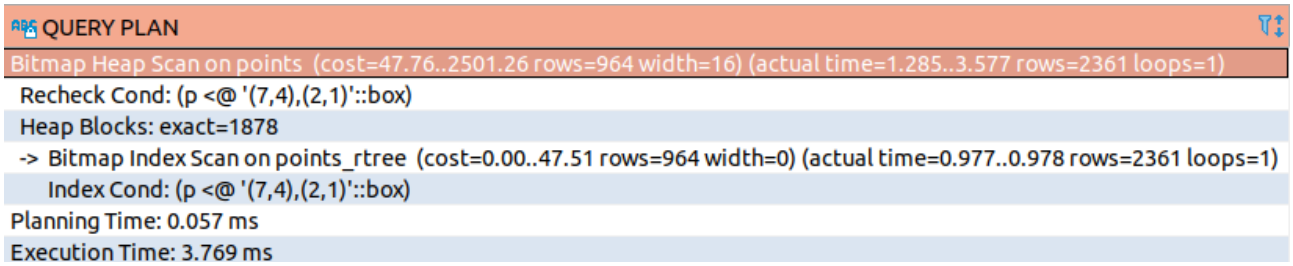
Slika 14: Rezultat izvršenja

Nakon toga, naredbom `create index points_rtree on points using gist(p);` kreiran je GiST indeks.

Da bi se kreirao indeks sa strukturom B stabla potrebno je da nad tipom podataka ključa budu definisani operatori poređenja. U slučaju tipa podataka point ovo nije slučaj pa samim tim nije ni moguće kreirati indeks sa strukturom B stabla. Operatori nad tipom podata point definisani su PostgreSQL operator klasom point_ops. Kao što je već rečeno GiST indeks služi kao interfejs za kreiranje indeksa sa proizvoljnom strukturom podataka, gde je potrebno definisati operatore koji će

se koristiti za kreiranje indeksa. U ovom slučaju, gde je korišćena point_ops operator klasa kreirani indeks ima strukturu R stabla.

Nakon kreiranog indeksa ukoliko se gornji upit ponovo izvrši dobija se sledeći rezultat:



Slika 15: Rezultat izvršenja

Na osnovu datog primera može se zaključiti da je korišćenje GiST indeksa u ovom slučaju potpuno opravdano s obzirom na to da je vreme izvršenja upita skraćeno približno 10 puta.

SP-GiST

SP-GiST je skraćenica za space-partitioned GiST. SP-GiST podržava particionisana stabla pretrage, koja olakšavaju razvoj širokog spektra različitih nebalansiranih struktura podataka, kao što su kvad-stabla, k-d stabla i radix stabla. Zajednička karakteristika ovih struktura je da više puta dele prostor za pretragu na particije koje ne moraju biti jednake veličine. Pretrage koje se dobro podudaraju sa pravilom particionisanja mogu biti veoma brze.

Ove popularne strukture podataka su prvobitno razvijene za korišćenje u glavnoj memoriji. U glavnoj memoriji, one su obično dizajnirane kao skup dinamički dodeljenih čvorova povezanih pokazivačima. Ovo nije pogodno za direktno skladištenje na disku, pošto ovi lanci pokazivača mogu biti prilično dugi što bi zahtevalo previše pristupa disku. Nasuprot tome, strukture podataka zasnovane na disku bi trebalo da imaju veliki fan-out da bi minimizirali I/O. Izazov kojim se bavi SP-GiST je mapiranje čvorova stabla pretrage na stranice diska na takav način da prilikom pretrage treba pristupiti manjem broju stranica na disku, čak i ako pretraga prolazi kroz mnogo čvorova.

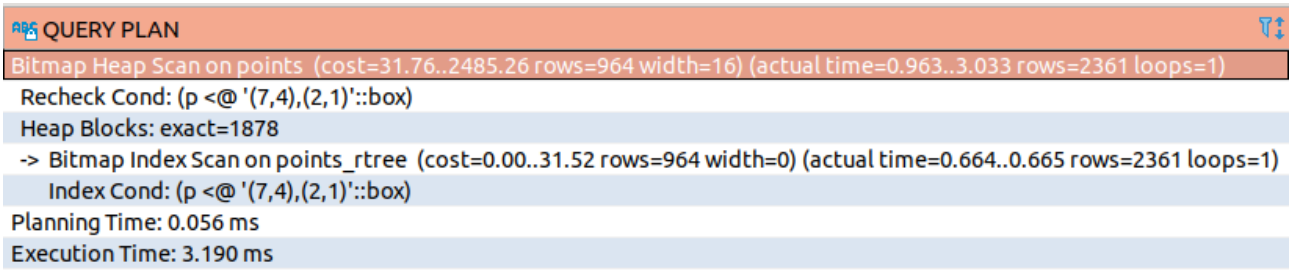
Kao i GiST, SP-GiST je namenjen da omogući razvoj prilagođenih tipova podataka sa odgovarajućim metodama pristupa.

Primer

Nad tabelom prikazanom u prethodnom primeru moguće je kreirati SP-GiST indeks komandom:

```
create index points_rtree on points using spgist(p);
```

Nakon toga, izvršenjem upita `explain analyze select * from points where p <@ box '(2,1),(7,4)';` dobija se sledeći rezultat:



Slika 16: Rezultat izvršenja

Slično kao I kod GiST-a operator klasaom se definiše koju će strukturu podataka imati kreirani indeks, u ovom sučaju koristi se operator klasa quad_points_ops pa je rezultujuća struktura podataka quad stablo.

U ovom primeru se vidi da SP-GiST indeks daje još bolje rezultate u odnosu na GiST indeks iz prethodnog primera.

GIN

GIN je skraćenica za generalizovani invertovani indeks(Generalized Inverted Index) . GIN je dizajniran za rukovanje slučajevima u kojima su stavke koje treba indeksirati složene vrednosti, a upiti kojima se obrađuje indeks treba da traže vrednosti elemenata koje se pojavljuju unutar složenih stavki. Na primer, stavke mogu biti dokumenti, a upiti mogu biti pretrage dokumenata koji sadrže određene reči.

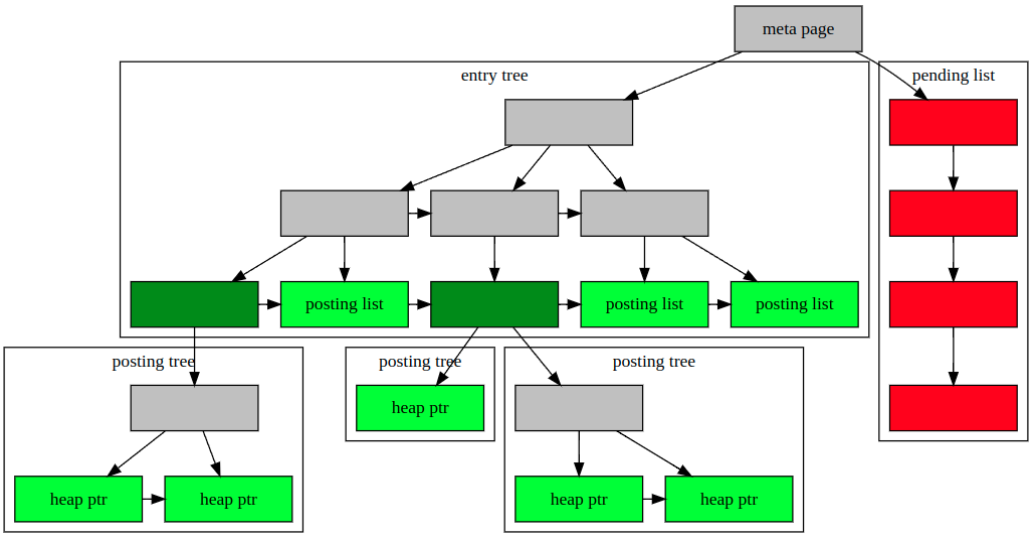
Reč stavka se odnosi na složenu vrednost koja treba da se indeksira, a reč ključ se odnosi na vrednost elementa. GIN uvek skladišti i traži ključeve, a ne vrednosti stavki same po sebi.

GIN indeks skladišti skup parova (ključ, lista objavljivanja), pri čemu je lista objavljivanja skup id-jeva redova u kojima se nalazi ključ. Isti id reda se može pojaviti u više listi objavljivanja, pošto stavka može da sadrži više od jednog ključa. Svaka vrednost ključa se čuva samo jednom, tako da je GIN indeks veoma kompaktan za slučajeve gde se isti ključ pojavljuje mnogo puta.

GIN je generalizovan u smislu da kod metode pristupa GIN-u ne mora da poznaje specifične operacije koje ubrzava. Umesto toga, koristi prilagođene strategije definisane za određene tipove podataka. Strategija definiše kako se ključevi izdvajaju iz indeksiranih stavki i uslova upita i kako da se utvrdi da li red koji sadrži neke od vrednosti ključa u upitu zaista zadovoljava upit.

Kao I kod GiST-a, jedna od prednosti GIN-a je što omogućava razvoj prilagođenih tipova podataka sa odgovarajućim metodama pristupa.

Interno, GIN indeks sadrži indeks B-stabla konstruisan nad ključeva, pri čemu je svaki ključ element jedne ili više indeksiranih stavki (član niza, na primer) i gde svaka tačka na stranici sa listom sadrži ili pokazivač na B-stablo pokazivača na heap („stablo objavljivanja“) ili jednostavnu listu pokazivača gomile („lista objavljivanja“) kada je lista dovoljno mala da stane u jedan indeksni skup zajedno sa ključnom vrednošću.



Slika 17: Interna struktura GIN indeksa

Primer

Za potrebe demostracije rada GIN indeksa iskorišćena je tabela ts u koju je upisan kompletan lorem ipsum tekst(94494 bajtova) podeljen na delove od po 100 karaktera zajedno sa ts vektorom :

```
conn = psycopg2.connect(
    host="localhost",
    database="dbms",
    user=[redacted]
    password=[redacted]

cond = True
count = 0
mycursor = conn.cursor()
while (cond):
    doc = text[count * 100: (count+1)*100]
    sql = f"INSERT INTO ts(doc) VALUES ('{doc}')"
    mycursor.execute(sql)
    count += 1
    if len(doc) == 0:
        cond = False
conn.commit()
```

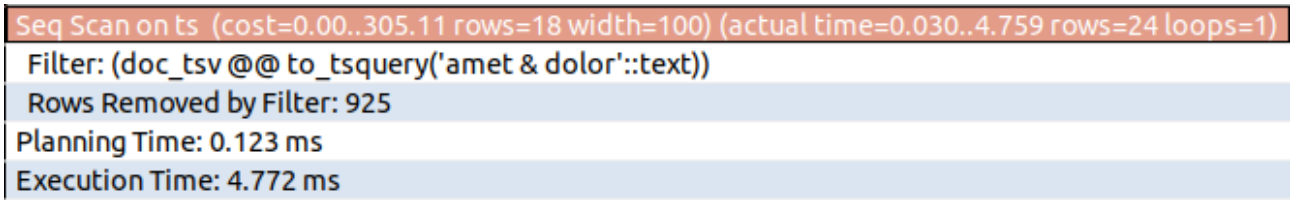
Slika 18: Primer koda

```
update ts set doc_tsv = to_tsvector(doc);
```

Nakon toga izvršen je sledeći upit:

```
explain analyze select doc from ts where doc_tsv @@ to_tsquery('amet & dolor');
```

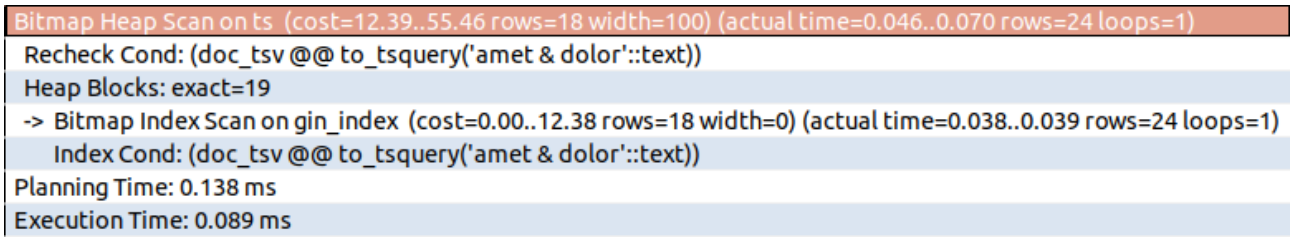
Rezultat izvršenja prikazan je na sledećoj slici:



Slika 19: Rezultat izvršenja

Nakon toga, naredbom `create index gin_index on ts using gin(doc_tsv);`Kreiran je GIN indeks.

Nakon kreiranog indeksa ukoliko se gornji upit ponovo izvrši dobija se sledeći rezultat:



Slika 20: Rezultat izvršenja

BRIN

BRIN je skraćenica od Block Range Indek. BRIN je dizajniran za rukovanje veoma velikim tabelama u kojima određene kolone imaju neku prirodnu korelaciju sa svojom fizičkom lokacijom unutar tabele. Opseg bloka je grupa stranica koje su fizički susedne u tabeli; za svaki opseg blokova, neke zbirne informacije se čuvaju u indeksu. Na primer, tabela u kojoj se čuvaju prodajne porudžbine prodavnice može da ima kolonu datuma kada je svaka porudžbina postavljena, a većinu vremena će se unosi za ranije porudžbine pojaviti i ranije u tabeli; tabela koja čuva kolonu sa poštanskim brojem može imati sve kodove za grad grupisane spontano.

BRIN indeksi mogu da zadovolje upite putem redovnog skeniranja indeksa bitmap-a i vratiće sve torke na svim stranicama u okviru svakog opsega ako su zbirne informacije koje indeks čuva u skladu sa uslovima upita. Izvršilac upita je zadužen za ponovnu proveru ovih tokova i odbacivanje onih koji ne odgovaraju uslovima upita, drugim rečima, ovi indeksi su sa gubitkom. Pošto je BRIN indeks veoma mali, skeniranje indeksa dodaje malo dodatnih troškova u poređenju sa sekvencijalnim skeniranjem, ali može izbeći skeniranje velikih delova tabele za koje se zna da ne sadrže odgovarajuće torke.

Specifični podaci koje će BRIN indeks čuvati, kao i specifični upiti koje će indeks moći da zadovolji, zavise od klase operatora izabrane za svaku kolonu indeksa. Tipovi podataka koji imaju linearni redosled sortiranja mogu imati klase operatora koje čuvaju minimalnu i maksimalnu vrednost unutar svakog opsega bloka, na primer; geometrijski tipovi mogu da čuvaju granični okvir za sve objekte u opsegu blokova.

Veličina opsega bloka je određena u vreme kreiranja indeksa parametrom skladištenja pages_per_range. Broj zapisa indeksa biće jednak veličini relacije u stranicama podeljenoj sa izabranom vrednošću za pages_per_range. Dakle, što je manji broj, indeks postaje veći (zbog potrebe da se uskladišti više unosa u indeks), ali u isto vreme sačuvani zbirni podaci mogu biti precizniji, i više blokova podataka može biti preskočeno tokom skeniranja indeksa.

Primer

Za potrebe demostracije rada BRIN indeksa iskorišćena je tabela coordinates u koju su upisane sekvencijalne vrednosti uz pomoć dve ugnježdene petrlje :

```
conn = psycopg2.connect(
    host="localhost",
    database="dbms",
    user='[redacted]
    password=[redacted]
mycursor = conn.cursor()
for i in range(1000):
    for j in range(10000):
        sql = f"INSERT INTO coordinates (x, y) VALUES ({i}, {j})"
        mycursor.execute(sql)
conn.commit()
```

Slika 21: Primer koda

Nakon toga izvršen je sledeći upit:

```
explain analyze select * from coordinates where x = 50 and y=5;
```

Rezultat izvršenja prikazan je na sledećoj slici:

```
Gather  (cost=1000.00..107748.10 rows=1 width=8) (actual time=37.092..254.240 rows=1 loops=1)
  Workers Planned: 2
  Workers Launched: 2
  -> Parallel Seq Scan on coordinates  (cost=0.00..106748.00 rows=1 width=8) (actual time=151.703..221.876 rows=0 loops=3)
       Filter: ((x = 50) AND (y = 5))
       Rows Removed by Filter: 3333333
Planning Time: 0.108 ms
JIT:
  Functions: 6
  Options: Inlining false, Optimization false, Expressions true, Deforming true
  Timing: Generation 1.296 ms, Inlining 0.000 ms, Optimization 1.101 ms, Emission 8.567 ms, Total 10.964 ms
Execution Time: 254.708 ms
```

Slika 22: Rezultat izvršenja

Nakon toga, naredbom `create index brin_index on public.coordinates using brin(x);` kreiran je BRIN indeks.

Nakon kreiranog indeksa ukoliko se gornji upit ponovo izvrši dobija se sledeći rezultat:

```
Bitmap Heap Scan on coordinates  (cost=12.03..41729.70 rows=1 width=8) (actual time=0.838..2.655 rows=1 loops=1)
  Recheck Cond: (x = 50)
  Rows Removed by Index Recheck: 18928
  Filter: (y = 5)
  Rows Removed by Filter: 9999
  Heap Blocks: lossy=128
  -> Bitmap Index Scan on brin_index  (cost=0.00..12.03 rows=28902 width=0) (actual time=0.057..0.058 rows=1280 loops=1)
       Index Cond: (x = 50)
Planning Time: 0.060 ms
Execution Time: 2.673 ms
```

Slika 23: Rezultat izvršenja

Hash

PostgreSQL uključuje implementaciju perzistentinh heš indeksa na disku, koji se u potpunosti mogu oporaviti nakon otkaza. Bilo koji tip podataka može biti indeksiran heš indeksom, uključujući tipove podataka koji nemaju dobro definisan linearni poredak. Heš indeksi čuvaju samo heš vrednost podataka koji se indeksiraju, tako da nema ograničenja za veličinu kolone podataka koja se indeksira.

Heš indeksi podržavaju samo indekse sa jednom kolonom i ne dozvoljavaju proveru jedinstvenosti.

Heš indeksi podržavaju samo operator =, tako da klauzule WHERE koje specificiraju operacije opsega neće moći da iskoriste prednosti heš indeksa.

Svaka heš indeksna torka skladišti samo heš vrednost od 4 bajta, a ne stvarnu vrednost kolone. Kao rezultat toga, heš indeksi mogu biti mnogo manji od B-stabala kada se indeksiraju duže stavke podataka kao što su UUID-ovi, URL-ovi, itd. Odsustvo vrednosti kolone takođe čini sva skeniranja heš indeksa “labavim”. Heš indeksi mogu da učestvuju u skeniranju indeksa bitmap-a i u skeniranju unazad.

Heš indeksi su najbolje optimizovani za SELECT i UPDATE upite sa velikim opterećenjima koja koriste skeniranje jednakosti nad velikim tabelama. U indeksu B-stabla, pretrage moraju da se spuštaju kroz stablo dok se ne pronađe lisna stranica. U tabelama sa milionima redova, ovo spuštanje može povećati vreme pristupa podacima. Ekvivalent lisne stranice u heš indeksu naziva se stranica segmenta. Nasuprot tome, heš indeks omogućava direktan pristup stranicama segmenta, čime se potencijalno smanjuje vreme pristupa indeksu u većim tabelama. Ovo smanjenje "logičkog I/O" postaje još izraženije na indeksima/podacima većim od shared_buffers/RAM-a.

Heš indeksi su dizajnirani da se nose sa neujednačenom distribucijom heš vrednosti. Direktan pristup stranicama segmenata funkcioniše dobro ako su heš vrednosti ravnomerno raspoređene. Kada umetanja znače da se stranica segmenta napuni, dodatne stranice za prekoračenje se vezuju za tu specifičnu stranicu segmenta, lokalno proširujući skladište za indeksne torke koji odgovaraju toj heš vrednosti. Kada se skenira heš segment tokom upita, moraju se skenirati sve stranice prekoračenja. Stoga bi neuravnoteženi heš indeks mogao biti gori od B-stabla u smislu broja potrebnih pristupa bloku za neke podatke.

Kao rezultat slučajeva prekoračenja, može se reći da su heš indeksi najpogodniji za jedinstvene, skoro jedinstvene podatke ili podatke sa malim brojem redova po heš segmentu. Jedan od mogućih načina da se izbegnu problemi je da se iz indeksa izuzmu veoma nejedinstvene vrednosti korišćenjem uslova delimičnog indeksa, ali to u mnogim slučajevima možda nije prikladno.

Kao i B-stabla, heš indeksi obavljaju jednostavno brisanje indeksnih torki. Ovo je odložena operacija održavanja koja briše indeksne torke za koje se zna da su bezbedni za brisanje (one čiji je LP_DEAD bit identifikatora stavke već postavljen). Ako se umetanjem otkrije da nema slobodnog prostora na stranici, pokušava se sa izbegavanjem kreiranja nove stranice prekoračenja pokušajem da se uklone mrtve indeksne torke.

Heš indeksi mogu da prošire broj stranica u segmentu kako broj indeksiranih redova raste. Mapiranje heš ključ-segment broja je izabrano tako da se indeks može postepeno proširivati. Kada treba da se doda novi segment u indeks, potrebno je da se „podeli“ tačno jedan postojeći segment, pri čemu će se neki od njegovih torki preneti u novi segment u skladu sa ažuriranim mapiranjem ključ-segment broja.

Proširenje se dešava u prvom planu, što bi moglo povećati vreme izvršenja za korisničke INSERT naredbe. Dakle, heš indeksi možda nisu pogodni za tabele kod kojih se brzo povećava broj redova.

Primer

Za potrebe demostracije rada Hash indeksa iskorišćen je sličan pristup kao I kod GIN indeksa, s tim što je tekst podeljen na delove od po 10 karaktera i nije korišćen ts vektor:

```
conn = psycopg2.connect(
    host="localhost",
    database="dbms",
    user="[redacted]
    password=[redacted]

cond = True
count = 0
mycursor = conn.cursor()
while (cond):
    doc = text[count * 10: (count+1)*10]
    sql = f"INSERT INTO hash_table(doc) VALUES ('{doc}')"
    mycursor.execute(sql)
    count += 1
    if len(doc) == 0:
        cond = False
conn.commit()
```

Slika 24: Primer koda

Nakon toga izvršen je sledeći upit: `explain analyze select * from hash_table where doc = 'Lorem ipsu';`

Rezultat izvršenja prikazan je na sledećoj slici:

```
Seq Scan on hash_table  (cost=0.00..170.51 rows=1 width=10) (actual time=0.014..1.299 rows=1 loops=1)
  Filter: (doc = 'Lorem ipsu'::text)
  Rows Removed by Filter: 9480
Planning Time: 0.061 ms
Execution Time: 1.311 ms
```

Slika 25: Rezultat izvršenja

Naredbom `create index hash_index on public.hash_table using hash(doc);` kreiran je Hash indeks.

Nakon kreiranog indeksa ukoliko se gornji upit ponovo izvrši dobija se sledeći rezultat:

```
Index Scan using hash_index on hash_table  (cost=0.00..8.02 rows=1 width=10) (actual time=0.015..0.016 rows=1 loops=1)
  Index Cond: (doc = 'Lorem ipsu'::text)
Planning Time: 0.072 ms
Execution Time: 0.030 ms
```

Slika 26: Rezultat izvršenja

Zaključak

U prvom delu ovog rada izložene su teoriske osnove vezane za internu strukturu I organizaciju indeska u modernim sistemima za upravljanje bazama podataka. Objašnjeno je kako funkcionišu primarni indeksi nad poljima koja pretstavljaju primarni ključ određene tabele, kako je moguće kreirati klastering indekse u slučajevima kada ključ nema jedinstene vrednosti, kao I način funkcionisanja sekundarnih ključeva u slučajevima kada nad datom tabelom već postoji primarni ključ.

Nakon toga pokazano je kako se radi uslojavanje ovakvih struktura I kako je moguće već poznate strukture podataka kao što su B I B+ stabla iskoristiti za formiranje višenivovskih indeksa. Na kraju objašnjene su metode višestrukog pristupa ključevima kao I neki drugi tipovi indeksa koji do tada nisu bili pomenuti.

U drugom delu rada akcenat je bio na tipovima indeksa koje pruža rešenje PostgreSQL baze podataka I tu su opisani indeksi tipa B-Tree, GiST, SP-GiST, GIN, BRIN I Hash. Za svaki od ovih tipova indeksa generisani su test podaci u realnom okruženju, kreiran je dati tip indeska I mereno je vreme izvršenja specifičnih upita sa korišćenjem indeksa I bez njega. Za svaki primer prikazane su razlike u vremenu izvršenja upita I time je demonstrirana opravdanost uvođenja datog tipa indeksa u konkretnom slučaju.

Korišćena literatura

1. R.Elmasri, S.Navathe, Fundamentals of Database, System, Addison Wesley, 2010, 6/e
2. https://www.postgresql.org/docs/current/index.html
3. Indexing in PostgreSQL-3(Hash): https://postgrespro.com/blog/pgsql/4161321
4. Indexing in PostgreSQL-4(Btree): https://postgrespro.com/blog/pgsql/4161516
5. Indexing in PostgreSQL-5(GiST): https://postgrespro.com/blog/pgsql/4175817
6. Indexing in PostgreSQL-6(SP-GiST): https://postgrespro.com/blog/pgsql/4175817
7. Indexing in PostgreSQL-7(GIN): https://postgrespro.com/blog/pgsql/4261647
8. Indexing in PostgreSQL-9(Hash): https://postgrespro.com/blog/pgsql/5967830
9. Sparse Index: http://mlwiki.org/index.php/Sparse_Index
10. Indexing in DBMS: https://www.javatpoint.com/indexing-in-dbms