UNIVERZITET U NIŠU
ELEKTRONSKI FAKULTET

Seminarski rad

Skladištenje i indeksi kod PostgreSQL baza

Student:

Dimitrije Mitić 822

Mentor:

Doc. dr Aleksandar Stanimirović

Sadržaj

1.Uvod

Pod pojmom baze podataka (***Database***) smatra se organizovana kolekcija povezanih podataka o nekoj organizaciji (pr. fakultet, biblioteka, video klub itd.) [1]. Potreba za bazama podataka javila se iz nedostataka do tada korišćenog konvencionalnog modela za skladištenje podataka (*File-based* sistem) u kome su aplikacije koje su vršile obradu podataka zavisile od samog načina na koji su podaci struktuirani, kao i zbog drugih nedostataka tipa: ograničena raspoloživost i nemogućnost zajedničkog korišćenja podataka, nedostatak mehanizma za oporavak i dr. Baza organizuje skup podataka na takav način da se isti može jednostavno: pretraživati, pregledati, sortirati, menjati i dr. Da bi se bazom moglo upravljati (definisanje, kreiranje, manipulisanje) postoji sistem za upravljanje bazama podataka (***DBMS***) koji vrši tu ulogu. DBMS je softver putem koga se vrši definisanje baze – specifikacija tipova podataka i ograničenja vezana za podatke koji će se skladištiti u bazi, konstruisanje baze – skladištenje podataka na neki spoljni medijum (hard disk, traka) koji je kontrolisan od DBMS-a, manipulacija bazom – obezbeđivanje funkcionalnosti koje se tiču vršenja upita nad bazom i vršenja update-a nad podacima, deljenje baze – mogućnost da više korisnika i aplikacija pristupa bazi istovremeno [2].

S obzirom na to da se sama baza podataka nalazi na eksternom skladištu (disk, traka) i da se podaci po potrebi prenose u glavnu memoriju radi obrade, DBMS vrši bitnu ulogu obezbeđivanja specifične strukture podataka i tehnike pretraživanja koje služe da ubrzaju pretragu kroz skladište i pribavljanje nekog konkretnog sloga. Pomoćne strukture podataka koje služe ubrzavanju pretrage su indeksi (***indexes***). Glavni fokus ovog rada jeste upoznavanje sa načinom i alternativama na koji DBMS organizuje podatke baze na disku (***file organization***) i upoznavanje sa indeksima (arhitektura, tipovi indeksa, implementacije) kao strukturama koje DBMS koristi radi optimizacije određenih operacija pribavljanja podataka, kao i konkretna primena indeksa u PostgreSQL bazi podataka.

Rad je organizovan u dve glavne celine. U prvoj je detaljno objašnjen način organizacije podataka na disku i indeksi, dok se drugi bavi konkretnom implementacijom ovih elemenata i primenama kod PostgreSQL baze.

2.Organizacija fajlova na disku i indeksi

Medijumi za skladištenje podataka kod kompjuterskih sistema su hijerarhijski organizovani i podeljeni na dve kategorije: primarno, sekundarno i tercijalno skladište. U kategoriju primarnih spadaju vrste skladišta kojima direktno procesor (CPU) može pristupati: glavna memorija (*main memory)* i keš memorija (*cache memory*). Ova vrsta skladišta se odlikuju velikom brzinom pristupa, ali ograničenim kapacitetom i gubitkom podataka nakon prestanka napajanja. U sekundarna i tercijalna, spadaju trajna skladišta koja se odlikuju velikim kapacitetom, dužim vremenom pristupa i manjom cenom. Većina baza podataka se trajno skladišti na sekundarnom skladištu, konkretno na **magnetnim diskovima** (*hard disk*) iz razloga što je generalno nemoguće držati veliku bazu u glavnoj memoriji, kao i zbog toga da je rizik kompletnog gubitka podataka znatno manji kod sekundarnog skladišta. Kad god je korisniku baze potreban neki podatak iz iste, on se najpre mora locirati na disku, zatim iskopirati u glavnu memoriju i na kraju, u koliko je podatak izmenjen, ponovo upisati na disk. S obzirom na to da je cena pristupa sekundarnom skladištu, znatno veća od cene pristupa podacima u primarnom skladištu, dizajner DBMS-a mora pažljivo odabrati strategiju organizacije podataka na disku, kojom bi bili, na efikasan način, zadovoljeni zahtevi korisnika baze.[2]

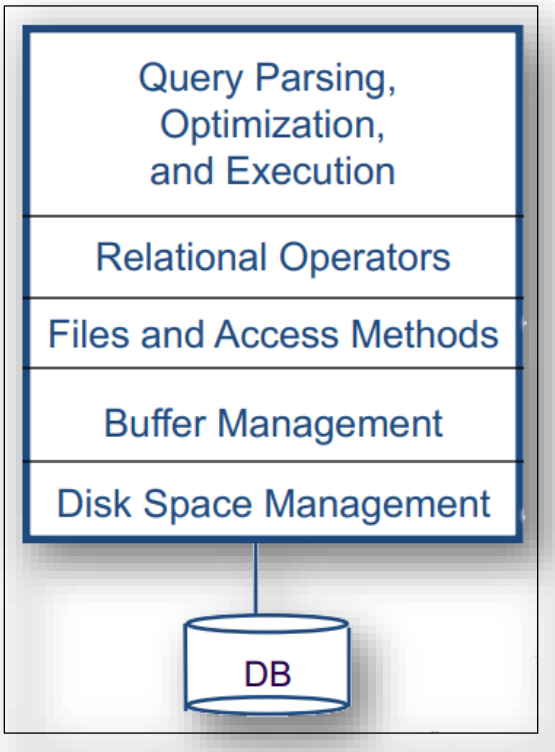
2.1 Blokovi, slogovi i fajlovi

Podaci su na disku organizovani u **blokove** (**stranice**, *pages*) fiksne veličine (512 do 8192 bajtova). Osnovna jedinica transfera podataka između glavne memorije i diska jeste blok (stranica). Magnenti disk spada u *random access* sekundarno skladište, što znači da se bilo kom bloku u disku može pristupiti po fiksnoj ceni, na osnovu njegove adrese. Lociranje i prenos susednih blokova sa diska je znatno jeftinije u odnosu na lociranje i prenos blokova koji nisu susedni. Iz ovog razloga je kod baza podataka potrebno minimizovati broj pristupa disku kod lociranja i transfera blokova sa diska do glavne memorije. Ovo je moguće postići odabirom odgovarajuće organizacije podataka baze na disku, tako što bi se „povezani“ podaci nalazili na susednim blokovima.[2][3]

Podaci u bazi se skladište u formi **slogova** (***records***) koji se sastoje od kolekcije povezanih vrednosti, gde svaka od njih predstavlja grupu bajtova koja predstavlja određenu vrednost atributa. Kolekcija slogova predstavlja **fajl** (***file***). Svaki slog u fajlu se sadrži svoj **rid**-a (record id) putem koga je moguće identifikovati adresu bloka na kome se slog nalazi. Svaki slog se mora dodeliti određenom bloku na disku, s obzirom da je blok osnovana jedinica transfera između diska i glavne memorije. Najčešće jedan blok sadrži više slogova. [3]

U nivovskoj arhitekturi DBMS-a, koja je data na Slici 1, postoje 3 hijerahijski poređana nivoa koji su zaduženi za komunikaciju sa eksternim uređajem. *Disk Space Manager* je najniži nivo koji direktno pristupa disku i zadužen je za alociranje i dealokaciju blokova na disku, kao i za samo preuzimanje i upis blokova. *Buffer Manager* nivo je zadužen za preuzimanje bloka koji mu obezbedjuje nivo ispod i za njegovo skladištenje u odgovarajući bafer glavne memorije. Bafer je podeljen na frejmove koji mogu biti zauzeti ili slobodni, i kada Buffer manager dobije zahtev od višeg nivoa za određenim slogom (na osnovi rid-a) on najpre pretražuje bafer i u koliko se blok

ne nalazi tamo, tek se onda pristupa disku. *Files and access methods* je nivo u kome je implementirana apstrakcija fajla kao kolekcije slogova. Ovo praktično znači da niži nivoi rade sa blokovima, dok File nivo radi sa slogovima i fajlovima. Glavne operacije koje se mogu vršiti nad fajlovima su: kreiranje i uklanjanje fajlova, dodavanje i brisanje slogova u konkretnom fajlu, kao i *scan* operacija kojom se obilaze svi slogovi u fajlu, jedan po jedan. Ovaj nivo takođe prati koje su stranice alocirane za koji fajl, kao i slobodan prostor u okviru samih stranica. [2][4]



Slika 1. Arhitektura DBMS-a [4]

2.2 Organizacija fajlova

Pod organizacijom fajlova se podrazumeva način na koji su slogovi i blokovi (stranice) poređane na spoljnom medijumu i način na koji su međusobno povezani. U čestom scenariju korišćenja baza, postoje neki uslovi upita koji se često koriste. Takođe, neki fajlovi u bazi su često podložni update operacijama (insert, delete, itd.) dok su drugi više statični. Dobra organizacija fajla treba služiti tome da se operacije koje se često primenjuju nad fajlom budu izvršavane što brže i efikasnije.

U nastavku ovog podpoglavlja biće opisane alternative koje se često koriste prilikom organizacije fajlova:

- ***Heap Files*** – Kod ove organizacije, slogovi se dodaju u fajl po redosledu njihovog dodavanja u bazu, što praktično znači da se novi slogovi dodaju na kraj fajla. Većina baza koristi ovakvu organizaciju u kombinaciji sa indeksima, koji se koriste radi efikasnije

pretrage. Heap organizacija omogućava brzo dodavanje podataka, što podrazumeva operacije kopiranja zadnjeg bloka fajla u glavnu memoriju, dodavanje sloga u blok, upis bloka na disk. Mana ovakve organizacije jeste neefikasna pretraga (linearna pretraga – blok po blok, dok se ne naiđe na traženi). [2]

- ***Sorted Files*** – Ovakva organizacija se dobija tako što se slogovi u fajlu sortiraju na osnovu vrednosti nekog polja (*ordering filed*). Ovakva organizacija olakšava pretragu na osnovu ordering field vrednosti (primenjuje se *binary search* tehnika) , kao i pribavljanje svih ili grupe sortiranih slogova, takođe na osnovu ordering field-a. Operacija dodavanja i brisanja je složenija u odnosu na heap organizaciju, zato što se mora očuvati uređenost. Takođe u koliko se zahteva pretraga na osnovu nekog polja koje nije ordering filed, sortirani fajl ne obezbeđuje nikakve prednosti tj. mora se primeniti linearna pretraga. [2]
- ***Hashed files –*** organizacija baziranja na tehnici *hashing*-a, koja omogućava brz pristup na osnovu jednakosti. Polje na osnovu koga se vrši poređenje se naziva *hash field,* u većini slučajeva *hash field* je istovremeno i ključ (*hash key*). Kada je potrebno pribaviti, dodati, izvršiti update, ili obrisati slog, hash funkcija određuje adresu traženog bloka, na osnovu *hash filed* vrdnosti. S obzirom da hash funkcija raspoređuje slogove uniformno po adresnom prostoru, ovakavim načinom skladištenja slogovi će biti smešteni na random memorijskim lokacijama. [2][4]
- ***Indexes –*** pomoćna struktura podataka čija je namena da ubrzaju pribavljanje slogova kod određenih vrsta pretraga. Ideja o indeksima je bazirana na ideji indeksa na kraju knjiga, gde se pored svakog termina nalaze strane na kojima se konkretni termin pojavljuje, tako da priliom pretrage za određenim terminom, nema se potrebe prolaziti koroz sam sadržaj knjige, stranu po stranu, već se na osnovu indeksa direktno pristupa strani koja sadrži termin. [2][3]

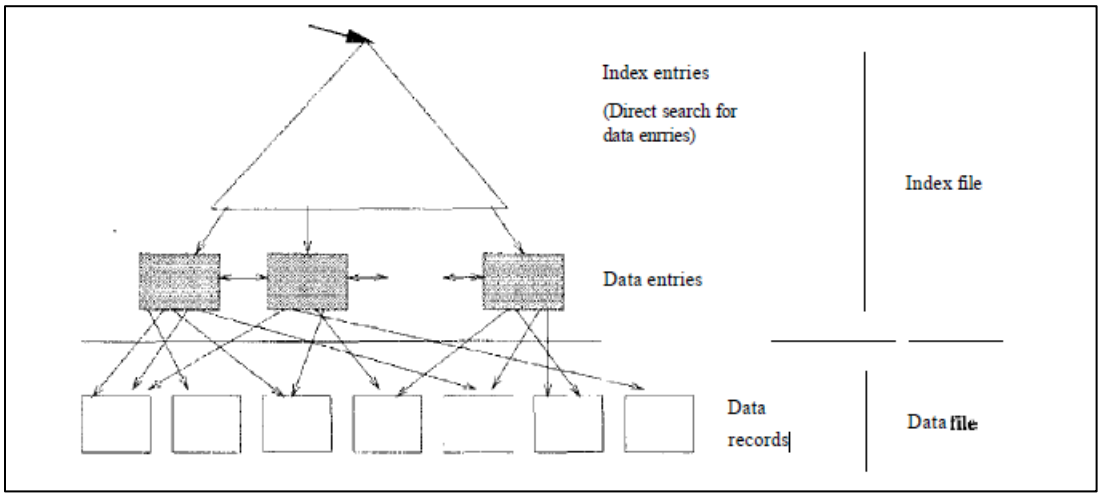
2.3 Indeksi, tipovi i implementacija

Indeks je struktura podataka koja se skladišti na disku i koja obezbeđuje alternativni i efikasni način za pristupanje slogovima fajla, bez da se utiče na samu organizaciju fajla na disku [3]. Uz pomoć indeksa je moguće na efikasan način pribavljanje sloga na osnovu polja indeksa (***index field*** *tj.* ***search key***) koji može biti napravljeno nad bilo kojim poljem (atributom) fajla. Pod pojmom ***data entrty*** podrazumeva se slog koji se čuva u indeksnom fajlu. Data entry za vrednost polja pretrage (search key) *k* označava se kao *k**. Uz pomoć data entry-a je moguće pristupiti samom slogu (direktno ili indirektno tj. preko pointera). Na slici 2 je dat prikaz indeksa. [3]

Postoje 3 alternative za sadržaj samog data entry-a *k**:

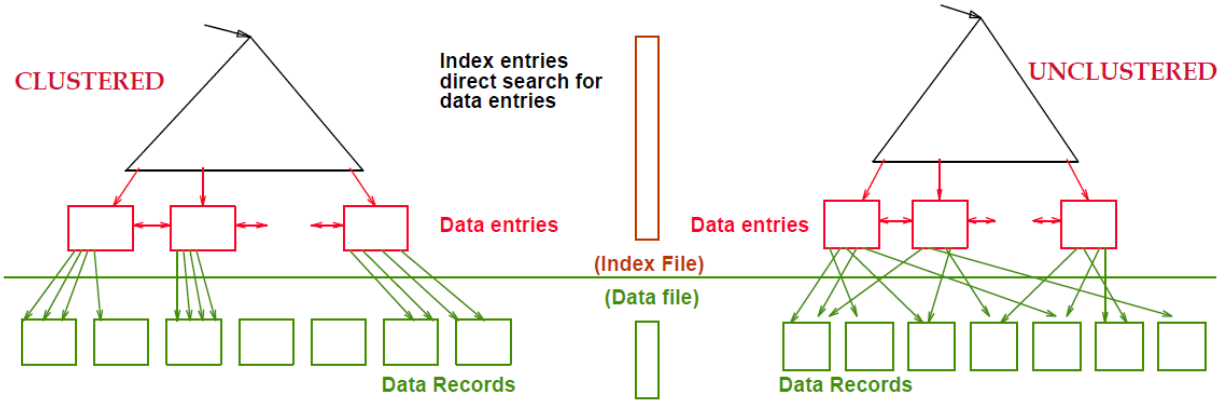
1. Data entry sadrži sam slog sa *saerch key* vrednošću *k*. Ovakav pristup se može smatrati specifičnom organizacijom samog fajla, koja se može koristiti kao zamena za sortirani fajl ili *heap* fajl. Nad nekom kolekcijom podataka može postojati samo jedan indeks ovog tipa, u suprotnom dolazi do dupliranja slogova i do potencijalne nekonzistentnosti.
2. Data entry je par (*k, rid*), gde je *rid* praktično adresa sloga čija je *search key*-a k. Ova alternativa obezbeđuje da se manje prostora izdvaja za data entry-e, što je bolja alternativa od alternative 1 u slučaju velikih slogova.

3. Data entry je par (*k, rid-listi*), gde *rid-list* označava listu rid-ova slogova koji imaju vrednost *search key*-a *k.* Ova alternativa je, kao i alternativa 2, nezavisna od same organizacije fajla. Takođe, ova alternativa obezbeđuje bolju iskorišćenost prostora od alternative 2, ali po cenu ne fiksne veličine *data entry*-a. Na slici 3, dat je



Slika 2 Indeks koji koristi alternativu 2 za data entry-e [3]

U koliko je data fajl organizovan tako da je raspored slogova isti, ili sličan kao i raspored slogova u indeks fajlu, onda je reč o klasterovanom indeksu (***clustered indexi***). Indeks koji je organizovan po alternativi 1 (data entry sadrži sam slog) je automatski i klasterovan. U koliko se radi o alternativama 2 i 3, indeks je klasterovan, samo u slučaju da je sam data file organizovan kao sortirani fajl po search key-u. S obzirom da su sortirani fajlovi neefikasni kada je reč o update-u, oni se retko koriste u praksi, tako da se pod klasterovanim indeksom podrazumeva alternativa 1. Klasterovani indeksi mogu značajno da ubrzaju pribavljanje opsega slogova, iz razloga što u klasterovanom indeksu susedni data entry-i ukazuju na susedna mesta u data fajlu, što znači da se određeni opseg slogova može pribaviti iz vrlo malog broja pristupa disku. Na slici 3 dat je prikaz klasterovanog i neklasterovanog indeksa indeksa, koji koristi alternativu 2 za data entry-e. [2]



Slika 3 Klasterovani I neklasterovani indeks [4]

Primarni indeks je indeks definisan nad sortiranim data fajlom, pri čemu je data file sortiran na osnovu **vrednosti primarnog ključa** tabele, koji je takođe i *search key*. Indeks fajl zauzima znatno manje prostora (blokva) na disku, iz dva razloga: broj data entry-a je manji u odnosu na broj slogova u data fajlu, takođe data entry obično zauzima manje prostora od prostora potrebnog za ceo slog. **Sekundarni indeks** se može definisati nad atributom koji je ključ kandidat i ima jedinstvenu vrednost u svakom slogu, ili može biti definisan nad ne ključ atributom (dva ili više slogova mogu imati istu vrednost ovog atributa). Takođe indeksi se mogu podeliti još na dve grupe: **gusti (*dense index*)** – postoji najmanje jedan *data entry* za svaku vrednost *search key*-a i **retki (*sparse index*)** – ne postoje data entry za svaku vrednost *search key*-a. [2][3]

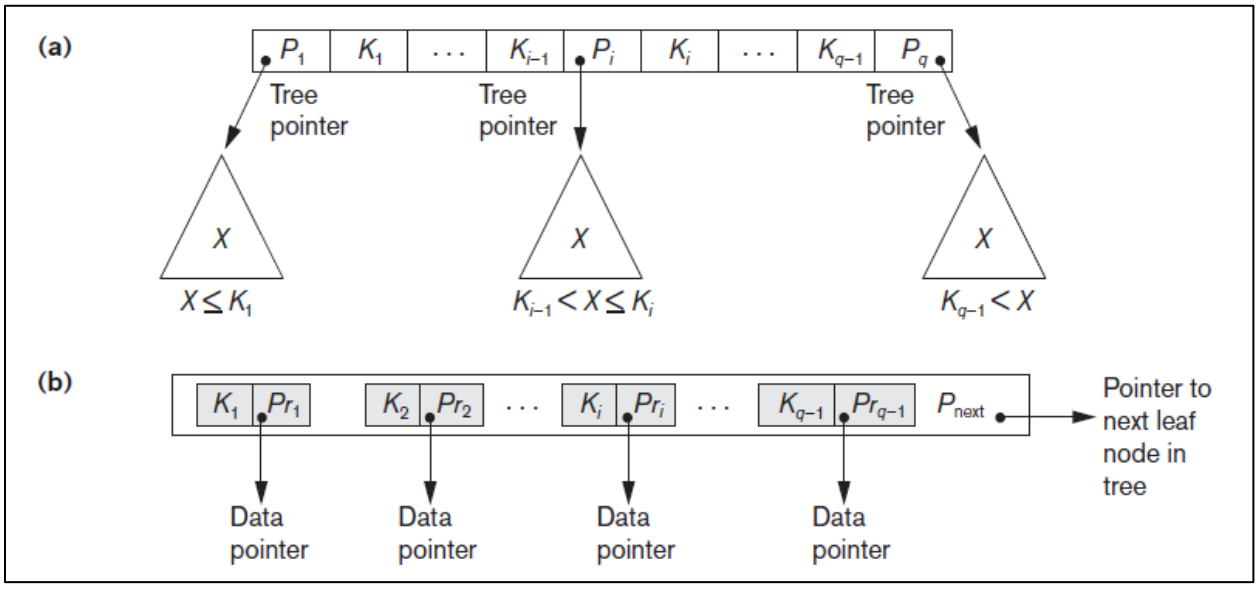
2.4 Struktura indeks fajla

Jedno od pitanja koje se nameće prilikom korišćenja indeksa, jeste na koji način je moguće organizovati sam indeks fajl, da bi se podaci na osnovu *search key*-a pribavljali što efikasnije (sa što manje pristupa disku). Dve glavna načina organizacije su: indeksiranje bazirano na *hashing*-u i ono bazirano na strukturi stabla (*tree-based indexinng*).

Kod ***hashed based*** indeksiranja, indeks fajl je organizovan u ***bucket***-e, pri čemu jedan *bucket* sadrži glavni blok i nijedan ili više *overflow* blokova povezanih u lanac strukturu. Određivanje bucket-a kome *data entry* pripada prilikom dodavanja ili pribavljanja sloga vrši se uz pomoć primene *hash* funkcije na polje pretrage (*search field*). Indeks baziran na *hashing*-u predstavlja jednonivovksi indeks fajl, koji je efikasan za pretraživanje po jednakosti, ali je neefikasan kada je potrebno vršiti pretragu na osnovu opsega vrednosti. [3]

Indeks može biti organizovan i kao struktura stabla (***tree-based***). Kod ovakve strukture, listovi stabla su *data entry,* sortirani po *search key*-u i ulančani (prethodni, sledeći), dok su čvorovi u ostalim, višim nivoima *index entry*-i koji služe za direkciju prilikom pretrage. Ovakva struktura čini pretragu baziranu na opsegu veoma efikasnom. Broj pristupa disku prilikom pretrage je jednak visini stabla (radi se o izbalansiranom stablu) plus broj ulančanih blokova koji sadrže tražene *data entry*-e. Struktura **B+ stabla**, koja se najčešće koristi za *tree-based* indekse, obezbeđuje da svi putevi od korena čvora (*root node*) do listova (*leaf nodes*) imaju istu dužinu,

tj. da stablo bude uvek visinski izbalansirano. Na slici 4 je dat prikaz strukture internog čvora B+ stabla koji se sastoji od *index entry*-a (a) i struktura leaf čvora, koji se sastoji od *data entry*-a. [3]



Slika 4 Struktura internog čvora (a) i leaf čvora (b) kod B+ tree indeksa [2]

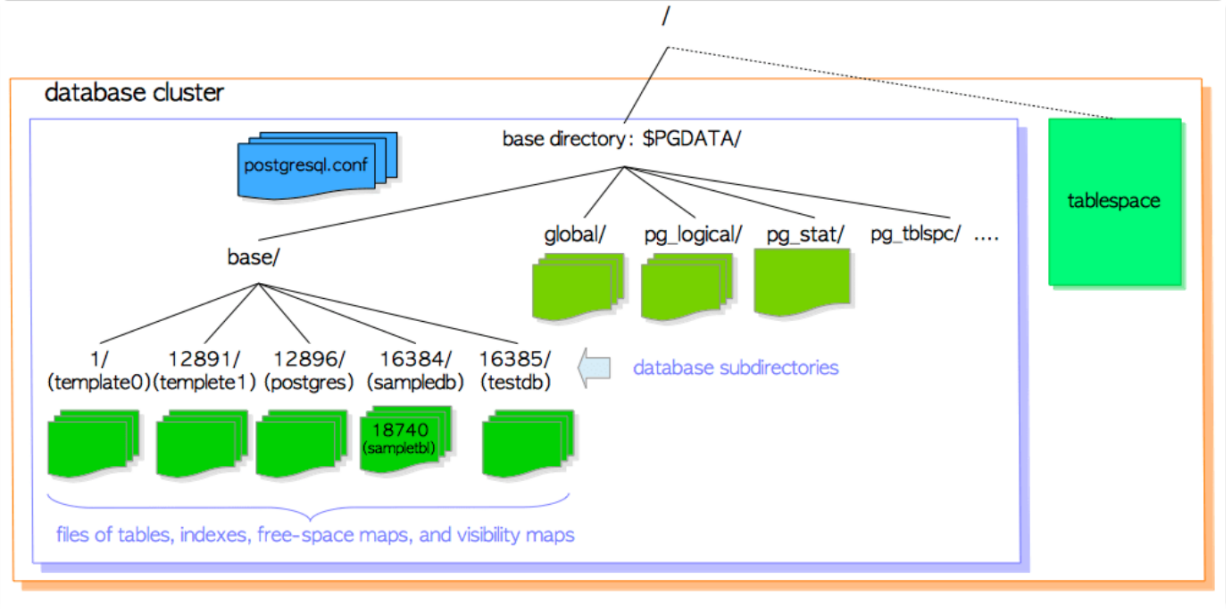
Sa slike se može videti da pointeri kod internih čvorov ukazuju na blokove koji takođe predstavljaju interne čvorove, dok pointeri kod *leaf* čvorova ukazuju na konkretne blokove *data* fajla. P_{next} pointer ukazuje na sledeći *leaf* čvor (blok). Na slici je izostavljen i pointer $P_{previos}$ koji ukazuje na prethodni *leaf* čvor. U praksi se *root* čvor smešta u *buffer* u glavnoj memoriji, zbog čestog pristupa istom. Pretraga podataka se odvija od *root* ka *leaf* čvorovima. Unos i ukljanjanje *data entry*-a se odvija na *leaf* čvorovima i ponekad su prilikom ovih operacija neophodne i promene na roditeljskim čvorovima naviše po stablu. Ovakva organizacija indeksa, znatno ubrzava pretragu kada se radi o veoma velikom broju *data entry*-a, po cenu potrebe za održavanjem jedne ovakve strukture prilikom operacija dodavanja i brisanje *data entry*-a i po cenu dodatnog prostora.

3. PostgreSQL

PostgreSQL je besplatan, *open-source,* objektno-relacioni RDBMS generlane namene. Postgre se smatra jednim od najnaprednijih *open-source* RDBMS-a danas. Neke od naprednih karakteristika ovog RDBMS-a su: mogućnost korisnika da definiše svoje tipove podataka (*user defined data types*), kreiranje procedura uz pomoć proceduralnog jezika PL/pgSQL, ili uz pomoć Python-a, Perl-a ili C jezika, mogućnost nasleđivanja tabela. Ove tri karakteristike su koncepti iz objektno orijentisane paradigme kojima je proširena relaciona paradigma, što dovodi do prevazilaženja razlika između ove dve paradigme i otvaranja novih mogućnosti kod dizajniranja baza podataka. U takođe bitne karakteristike PostgreSQL-a spadaju i: ugnježdene transakcije (*nested transactions*), *tablespace* – mogućnost da administrator sam definiše gde će se koji fajlovi baze skladištiti, *Multi Version Concurrency Controll (MVCC)* – sofisticirani način zaključavanja tj. kontrola konkurentnog pristupa redovima, asinhrona replikacija baze i dr. Jedan od razloga za popularnost ovog RDBMS-a je i u tome što je dizajniran tako da bude lako nadogradiv, tj. ukoliko mu je potrebno, korisnik sam može, kroz *plugin*-e, nadograđivati sistem. [6][7]

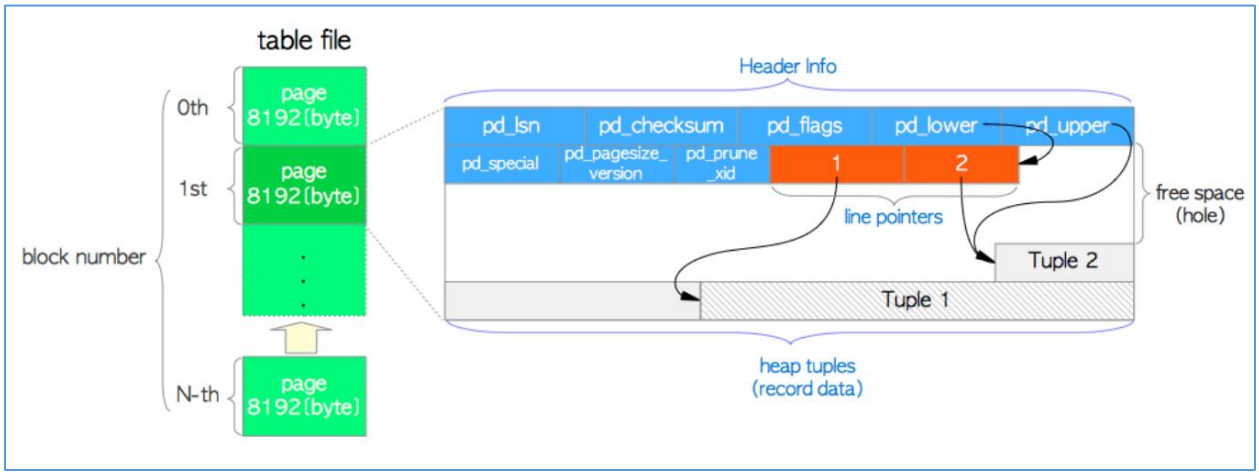
3.1 Organizacija fajlova kod PostgreSQL-a

Skup baza za koje je direktno zadužen konkretni PostgreSQL server, naziva se ***database cluster***. Kod PostgreSQL-a sama baza se sastoji od kolekcija ***database object***-a, u šta spadaju strukture podataka koje sadrže same podatke (tabele) ili reference na njih (indeksi, sekvence, pogledi itd.) Svakom *database object*-u je dodeljen jedinstveni dentifikator dužine 4B – ***object identifier (OID).*** Sam klaster predstavlja jedan direktorijum, tzv. ***base directory,*** čiji se put čuva u promenljivoj okruženja $PGDATA. U okviru ovog direktorijuma, nalazi se poddirektorijum *base,* koji sadrži po jedan konkretan poddirektorijum za svaku bazu koja se nalazi u klasteru, pri čemu ti poddirektorijumi baza nose naziv prema *OID*-u koji označava konkretnu bazu (pr. ako se radi o bazi čiji je *OID* 1476, sami fajlovi tabela i indeksa će se nalaziti na sledećem putu $PGDATA/base/1476/). Za svaku tabelu i indeks, u okviru neke baze, postoji odgovarajući fajl koji je takođe imenovan po *OID*-u koji je dodeljen toj tabeli odnosno indeksu. U koliko veličina fajla prelazi 1Gb, kreira se novi fajl sa istim nazivom kome se samo pridodaje odgovarajući broj kao sufiks, nakon čega se novi blokovi pamte u tom novom fajlu, dok njegova veličina takođe ne dostigne 1Gb. [8]



Slika 5 Organizacija database cliuster-a kod PostgreSQL-a [8]

Fajlovi kojima su predstavljene tabele i indeksi kod PostgreSQL-a, se sastoje od blokova čija je default veličina 8KB. Veličina bloka se može menjati (maksimalna vrednost je 32KB) ali se to ne preporučuje. Slogovi, koji se nalaze u ovim fajlovima, su organizovani u obliku Heap strukture čiji je prikaz dat na slici 6. Ovakvu strukturu je moguće pretraživati sekvencijalno (red po red, blok po blok), ili uz pomoć pomoćne strukture - indeksa.



Slika 6 Heap organizacija fajla kod PostgreSQL-a [8]

3.2 Tipovi indeksa kod PostgreSQL-a

Radi efikasnije pretrage velikih tabela, na osnovu često korišćenih parametara upita, PostgreSQL koristi odvojenu strukturu podataka tj. indekse. Sam Postgre daje mogućnost korišćenja različitih tipova indeksa, koji koriste različite algoritme za pretragu, koji su najbolje prilagođeni određenim vrstama upita. Koju vrstu indeksa ćemo upotrebiti, zavisi od tipa kolone na osnovu

koje pravimo upit, kao i od operatora koje koristimo u upitu. Neki od najkorišćenijih vrsta indeksa, koji su podržani kod PostgreSQL-a su:

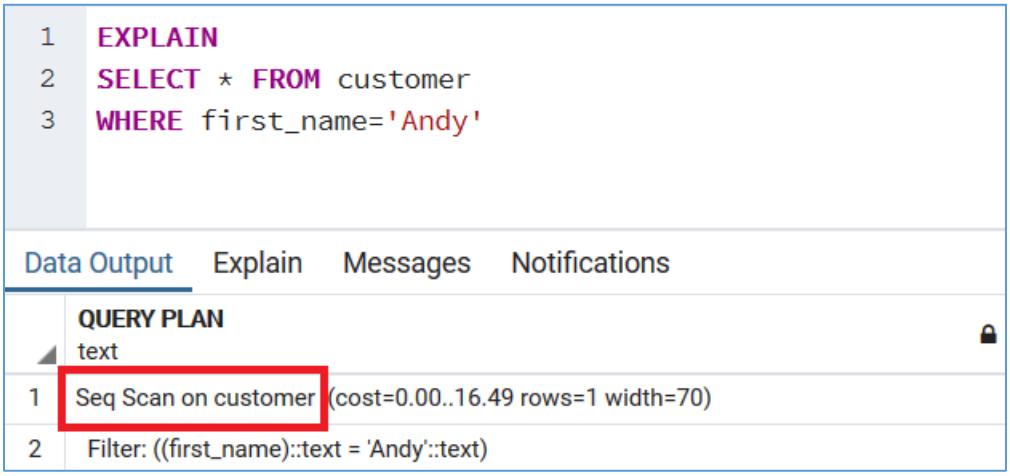
- ***Hash indexes*** - Ova vrsta indeksa se koristi za najjednostavnije upite koji su bazirani na proveri jednakosti. Planer upita će uzeti u obzir korišćenje hash indeksa, u koliko u upitu naiđe na uslov u kome je navedena indeksirana kolona i operator „=“. Ovakva vrsta indeksa organizovana je uz pomoć bucket-a, u kojima se na osnovu rezultata hash funkcije primenjene nad indeksiranom kolonom, smeštaju torke, po principu objašnjenom u poglavlju 2.4. Bitno je naglasiti da su hash indeksi kod PostgreSQL-a znatno slabiji od B-tree indeksa u pogledu različitosti operatora koji se mogu koristiti u upitima. [9]
- ***B-tree indexes*** – Ova vrsta indeksa je bazirana na samobalansirajućoj strukturi stabla, koja omogućava efiaksnu pretragu, insertovanje i bisanje podataka iz nje, po logaritamskom vremenu. Ovo je deafult indeks koji će PostgreSQL kreirati kada mu se izda CREATE INDEX komanda, u koliko pri tome se ne navede željeni tip. Ova vrsta indeksa podržava razne vrste pretraga baziranih na: jednakosti, nejednakosti, opsegu (BETWEEN operator, >=, <=, >, <) , kao i u nekim upitima koji koriste patern matching. Indekse ovog tipa, PostgreSQL automatski kreira prilikom definisanja primarnog ključa ili kolone sa unique constraint-om. [9]
- ***GIN (Generalized inverted indexes) –*** Definiše se nad kolonama koje nisu atomične, tj. sastoje se od više elemenata. Prilikom indeksiranja, ono se ne vrši nad samim, kompozitnim, atributom, već nad njegovim elementima. GIN indeks se satoji od (*key, list*) parova, gde je *key* vrednost elementa kompozitnog atributa, dok je *list* set ID-a redova u kojima se dati element pojavljuje, pri čemu je moguće da isti red bude povezan sa više ključeva. Svaka vrednost ključa se pamti samo jednom, što ovu strukturu čini prilično kompaktnom. Glavna primena ove vrste indeksa je kod Full-text pretraga, kod kojih je tip kolone dokument, a elementi od kojih se sastoji su reči (lekseme). Nedostatak GIN-a je u tome da je unos novih podataka vremenski zahtevan proces, iz razloga što se određeni dokument sastoji od velikog broja reči, koje je potrebno indeksirati. [9][10]
- ***GiST (Generalized search tree) –*** Potreba za ovom vrstom indeksa, javila se iz razloga što se u okviru tabele mogu pamtiti kompleksni tipovi podataka kao što su: geo podaci, slike, ili tekst. Nad ovakvim podacima je nedovoljno korišćenje standardnih operatora koju pružaju B-tree indeksi. GiST indeksi omogućuju organizovanje podataka kompleksnog tipa u strukturu balansiranog stabla, kao i definisanje metoda koje će koristiti operatori prilikom pristupa ovom stablu. Važno je naglasiti, da GiST ne predstavlja jedan konkretan tip indeksa, već svojevrsni framework, koji omogućava implementaciju različitih strategija indeksiranja. GiST je veoma koristan kod indeksiranja geometrijskih tipova podataka (tačka, poligon, itd), ali i kod full tekst pretrage. [11]

3.3 Korišćenje PostgreSQL indeksa

U ovom delu poglavlja će kroz konkretne primere, biti pokazan uticaj korišćenja indeksa na samu pretragu. Kao primer uzeta je tabela, čiji su prvih 10 redova dati na slici 7. Prilikom kreiranja jedne ovakve tabele, PostgreSQL implicitno kreira indeks nad kolonom primarnog ključa. U koliko bismo sada izdali upit za nalaženje reda koji za polje first_name ima npr vrednost „Andy“, planer upita bi prvo pogledao da li nad ovom kolonom postoji indeks, u koliko ovaj ne postoji, kao što je to trenutno slučaj, sekvencijalna pretraga redova se vrši (red po red, blok po blok) sve dok se na naiđe na traženu vrednost, ili se ne obišu svi slogovi. U koliko na jedan ovakav upit dodamo opciju EXPLAIN, dobijamo potvrdu potvrdu da će PostgreSQL obaviti sekvencijalnu pretragu. Prikaz ovog upita i rezultata dat je na slici 8.

customer_id [PK] integer	store_id smallint	first_name character varying (45)	last_name character varying (45)	email character varying (50)
1	1	Mary	Smith	mary.smith@sakilacustomer...
2	1	Patricia	Johnson	patricia.johnson@sakilacust...
3	1	Linda	Williams	linda.williams@sakilacusto...
4	2	Barbara	Jones	barbara.jones@sakilacusto...
5	1	Elizabeth	Brown	elizabeth.brown@sakilacust...
6	2	Jennifer	Davis	jennifer.davis@sakilacustom...
7	1	Maria	Miller	maria.miller@sakilacustome...
8	2	Susan	Wilson	susan.wilson@sakilacustom...
9	2	Margaret	Moore	margaret.moore@sakilacust...
10	1	Dorothy	Taylor	dorothy.taylor@sakilacusto...
11	2	Lisa	Anderson	lisa.anderson@sakilacusto...

Slika 7 Test tabela



Slika 8 Sekvencijalana pretraga

Za kreiranje indeksa nad konkretnom kolonom tabele, koristi se komanda CREATE INDEX. Prilikom izdavanje ove komande, pored navođenja neophodnih polja, kao što su naziv tabele i kolone, moguće je još navesti i tip indeksa koji se želi kreirati (btree, hash,…), u koliko se tip ne

navede, Postgre automatski kreira **btree** indeks. Na slici 9. Dat je prikaz komande za kreiranje indeksa nad kolonom *last_name* testne tabele.

```
CREATE INDEX idx_first_name
ON customer(first_name)
```

Slika 9 Kreiranje indeksa

U koliko sada pokušamo sa malopređašnjom pretragom sa slike 8 na osnovu kolone *first_name*, dobijamo rezultat prikazan na slici 10. Sa slike se vidi da planer upita koristi kreirani indeks, što znatno ubrzava cenu pretrage, koja se računa na osnovu broja pribavljenih blokova. Ubrzanje proizilazi iz toga što kod btree indeksa broj pribaljanih blokova je jednak visini samog stabla + 1 (dodatni pristup za pribavljanje bloka u kome se nalazi traženi slog).

	QUERY PLAN text
1	Index Scan using idx_first_name on customer (cost=0.28..8.29 rows=1 width=70)
2	Index Cond: ((first_name)::text = 'Andy'::text)

Slika 10 Pretraga sa indeksom

U nekim slučajevima je potrebno da se indeks napravi samo za neke delove tabele, kako bi se poboljšala efikasnost tog indeksa smanjivanjem njegove veličine. Primer ovoga bi bio, kada bi za našu tabelu *customer*, bilo potrebno da se ubrza pretraga po polju *last_name*, ali pri čemu nas interesuju samo aktivne mušterije (vrednost polja *active* im je postavljena na 1). Kreiranje ovakvog indeksa prikazano je na Slici 11, pri čemu se ovakva struktura naziva **parcijalni indeks** (*partial index*). Prilikom kreiranja, WHERE odrednica se koristi da bi se ograničio deo tabele na koji će se indeks odnositi. Da bi planer upita upotrebio indeks kod pretrage, potrebno je da u upitu budu navedene obe odrednice (*last_name* i *active*), u koliko bi pokušali pretragu samo po polju *last_name*, planer bi upotrebio sekvencijalnu pretragu.

```
CREATE INDEX idx_last_name
ON customer(last_name)
WHERE active=1
```

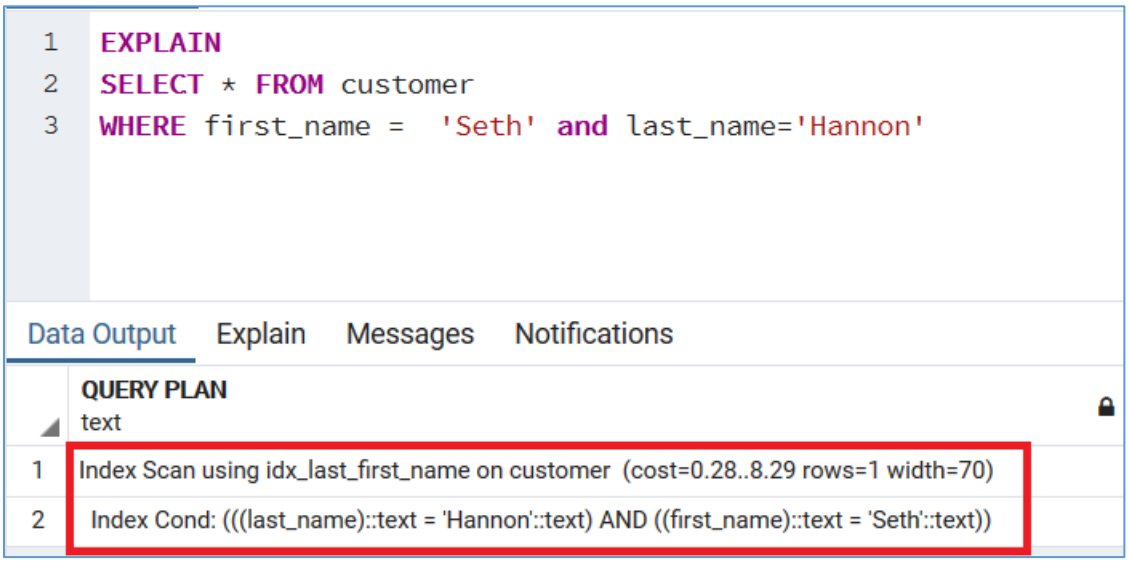
Slika 11 Parcijalni indeks

PostgreSQL omogućava i kreiranje indeksa nad više kolona tj. kreiranje **kompozitnih indeksa** (*multicolumn index*). Ovakvi indeksi se koriste za ubrzavanje pretrage na osnovu kolona koje se zajedno često koriste u upitima. Primer jednog ovakvog indeksa dat je na slici 12, gde smo najpre obrisali prethodno kreirane pojedinačne indekse nad kolonama *first_name* i *last_name* korišćenjem DROP komande, a zatim kreirali jedan kompozitni indeks.

```
DROP INDEX idx_first_name;
DROP INDEX idx_last_name;
CREATE INDEX idx_last_first_name ON
customer(last_name,first_name)
```

Slika 12 Kompozitni indeks

Uz pomoć ovako kreiranog indeksa, moguće je brzo pretraživanje tabele na osnovu polja *last_name* ili na osnovu oba polja (*last_name* i *first_name*) bez obzira na redosled navođenja, ali u koliko bi trebalo obaviti pretragu samo na osnovu polja *first_name*, planer bi krenuo u sekvencijalno traženje. Razlog za ovo je taj da su, u kreiranom indeksu, slogovi sortirani najpre po prvoj navedenoj koloni (*last_name*), a tek onda po drugoj (*first_name*). Na slici 13 dat je primer korišćenja kompozitnog indeksa.



Slika 13 Pretraga sa kompozitnim indeksom

U upitu na slici 13 korišćen je AND operator, u koliko bismo ga zamenili sa OR, kreirani indeks nam ne bi pomogao u traženju. Da bi ubrzali pribavljanje slogova kod jednog takvog upita, bilo bi potrebno kreirati indeks i nad kolonom *last_name*. U slučajevima izvršenja upita koji uključuje kombinovanje više kolona, nad kojima su definisani indeksi, Postgre će najpre pretražiti svaki indeks pojedinačno, nakon svakog prolaza će napraviti **bitmap**-u u memoriji kojom će biti označeni redovi koji su rezultat pretrage kroz konkretan indeks. Na kraju, Postgre izvšava određene bitwise operacije (AND, OR) nad bitmapama, nakon čega se dobijaju samo redovi koji su zadovoljili date uslove.

4. Zaključak

Indeksi su pomoćne fajl strukture (pamte se uz *database* fajlove), koje omogućavaju brzu pretragu podataka na osnovu nekog atributa nad kojim je indeks napravljen, nalik na indekse na kraju knjiga. Alaternativa korišćenju indeksa, bi bila sekvencijalna pretraga blokova i redova, koja jeste izvodljiva za neke manje tabele, ali u slučaju velikih, ovo je vremenski jako zahtevan proces koji zahteva veliki broj pristupa spoljnoj memoriji i prenošenje podataka iz iste u glavnu. Iz ovog razloga svi savremeni RDMBS-i omogućuju mehanizme za kreiranje i korišćenje različitih vrsta indeksa. U zavisnosti od tipa atributa nad kojim želimo da definišemo indeks i ubrzamo pretragu, postoje različiti tipovi indeksa koji ubrzavaju određene vrste pretraga. Neki omogućuju efikasnu pretragu na osnovu jednakosti, neki na osnovu opsega ili pripadnosti nekom skupu vrednosti, drugi pak vrše efikasnu full-text, ili pretragu geo podataka. Svima im je zajedničko to da ubrzavaju pretragu na račun izvesnih troškova održavanja. Ovo praktično znači da će se upiti koji uključuju naredbe SELECT i WHERE izvršavati znatno brže, dok će se naredbe koje se odnose na update redova, kao što su INSERT ili UPDATE, izvšavati nešto sporije, zato pto je potrebno, pored tabela, i ažuriranje pomoćnih struktura. Postoje i slučajevi kada nije poželjna upotreba indeksa, kao kod malih tabela ili onih koje su često izložene promenama.

5. Literatura

[1] https://www.tutorialspoint.com/dbms/dbms_overview.htm

[2] Ramez Elmasri and Shamkant Navathe. 2010. *Fundamentals of Database Systems* (6th. ed.). Addison-Wesley Publishing Company, USA.

[3] Raghu Ramakrishnan and Johannes Gehrke. 2002. Database Management Systems (3rd. ed.). McGraw-Hill, Inc., USA.

[4] https://www2.cs.duke.edu/courses/fall16/compsci516/Lectures/Lecture-5-6-Storage-Index.pdf

[5] https://www.tutorialspoint.com/dbms/dbms_indexing.htm

[6] https://www.postgresql.org/docs/10/intro-whatis.html

[7] https://www.tutorialspoint.com/postgresql/postgresql_overview.htm

[8] http://www.interdb.jp/pg/pgsql01.html

[9] https://www.postgresql.org/docs/10/indexes-types.html

[10] https://habr.com/en/company/postgrespro/blog/448746/

[11] https://postgrespro.com/blog/pgsql/4175817