Univerzitet u Nišu,

Elektronski fakultet Niš

Logo

Description automatically generated

*Seminarski rad*

Sistemi za upravljanje bazama podataka

**INTERNA STRUKTURA I ORGANIZACIJA INDEKSA  
KOD MICROSOFT SQL SERVER-A**

Mentor: Student:

Aleksandar Stanimirović Andrija Milosavljević 1481

Niš, april 2023. godine

**Sadržaj**

[**1.** **Uvod** 3](#_Toc132638240)

[**2.** **Interna struktura i organizacija indeksa** 5](#_Toc132638241)

[**2.1.** **Klasifikacija indeksa** 6](#_Toc132638242)

[**2.2.** **Struktura indeksa** 7](#_Toc132638243)

[**3.** **Interna struktura i organizacija indeksa kod MS SQL Server-a** 10](#_Toc132638244)

[**3.1.** **Klasterovani indeksi** 12](#_Toc132638245)

[**3.2.** **Neklasterovani indeksi** 14](#_Toc132638246)

[**3.3.** **Jedinstveni indeksi** 16](#_Toc132638247)

[**3.4.** **Filtrirani indeksi** 17](#_Toc132638248)

[**3.5.** **Columnstore indeksi** 18](#_Toc132638249)

[**3.6.** **Memorijski optimizovani indeksi** 21](#_Toc132638250)

[**3.7.** **XML indeksi** 23](#_Toc132638251)

[**3.8.** **Prostorni indeksi** 25](#_Toc132638252)

[**3.9.** **Full-text indeksi** 26](#_Toc132638253)

[**4.** **Zaključak** 27](#_Toc132638254)

[**5.** **Literatura** 28](#_Toc132638255)

# **Uvod**

Sistemi za upravljanje bazama podataka (eng. Database Management System - DBMS) skladište ogromnu količinu podataka, i podaci moraju postojati i nakon završetka izvršavanja programa. Zbog toga se podaci skladište na spoljnjim uređajima za skladištenje, kao što su diskovi i trake, i preuzimaju u glavnu memoriju kada su neophodni za obradu. Stranica (eng. Page) je jedinica informacija koja se čita sa diska ili upisuje na disk. Veličina jedne stranice kod DBMS-a je obično 4KB ili 8KB [1].

Diskovi su najvažniji spoljnji uređaj za skladištenje podataka. Oni nam omogućavaju da preuzmemo bilo koju stranicu po fiksnoj ceni. Jedan od načina da se smanji vreme čitanja podataka sa spoljnih uređaja je da se podaci pročitaju u onom redosledu u kojem su fizički skladišteni, umesto nasumično. Trake su spoljnji uređaji koji nas primoravaju da čitamo podatke u sekvenci. One su jeftinije od diskova, međutim, zbog sporijeg vremena pristupa i sekvencijalne prirode pristupa, trake nisu pogodne za transakcijske ili aplikacije u realnom vremenu. Uglavnom se koriste za arhiviranje podataka, tj. za skladištenje velikih količina podataka kojima se retko pristupa i koji ne zahtevaju brzo vreme pristupa.

Podaci se često čitaju ili upisuju na disk, što može biti veoma spor proces. Troškovi I/O operacija (ulazak podataka sa diska u glavnu memoriju i izlazak podataka iz memorije na disk) su znatno veći nego troškovi tipičnih operacija baze podataka, zbog toga se sistemi za upravljanje bazama podataka pažljivo optimizuju kako bi se minimizovali ovi troškovi. Cilj optimizacije je smanjenje troškova I/O operacija kako bi se postigla što brža i efikasnija obrada podataka [1].

U sistemima za upravljanje bazama podataka, podaci su organizovani kao kolekcije slogova, ili datoteke, i svaka datoteka se sastoji od jedne ili više stranica. Svaki slog u datoteci ima jedinstveni identifikator, koji se naziva ID sloga (eng. Record ID), ili skraćeno *rid*. Kada je poznat ID sloga, DBMS koristi tu informaciju da bi pronašao tačnu stranicu na disku koja sadrži traženi slog. To se obično radi pomoću B-stabla ili drugih algoritama za pretraživanje, koji omogućavaju brzo pronalaženje traženog sloga u datoteci. Ovaj proces identifikacije stranice na disku je ključan za efikasno upravljanje velikim količinama podataka u DBMS-u.

Za efikasno skladištenje i preuzimanje podataka DBMS koristi sloj datoteka i metoda pristupa (eng. Files and Access Methods), u nastavku ćemo ga nazivati samo sloj datoteka, koji organizuje podatke kako bi podržao brz pristup željenim podacima. Ovaj sloj koristi različite metode pristupa kako bi dobio podatke iz baze, kao što su indeksi, heširanje i druge tehnike koje omogućavaju sistemu da brzo locira i pronađe podatke na osnovu specifičnih kriterijuma.

Uzmimo za primer da imamo datoteku u kojoj se nalaze svi zaposleni u nekoj kompaniji, od kojih svaki sadrži ime, starost i platu. Ako želimo da dobijemo listu najstarijih zaposlenih, datoteku bismo morali da sortiramo po starosti, a operacija sortiranja je jako skupa ako se datoteka često menja. Ako želimo da nađemo zaposlene koji zarađuju više od 2000$, moramo da skeniramo celu datoteku da bismo pronašli takve zaposlene. Tehnika koja se zove indeksiranje nam omogućava brži pristup određenom podskupu slogova tako da se podaci sortiraju i organizuju na osnovu određenih ključeva, a zatim se stvara struktura indeks koja pokazuje na tačne lokacije na disku. Indeksiranjem omogućavamo DBMS-u da brzo pristupa slogovima pomoću ključeva, bez potrebe za pretraživanjem cele datoteke.

S obzirom da ćemo u ovom radu najviše pažnje posvetiti indeksima, u slojevitoj arhitekturi DBMS-a najbitniji slojevi su uokvireni crvenim pravougaonikom na *slici 1.1*. Već smo pričali o sloju datoteka, a naredni sloj je bafer menadžer (eng. Buffer Manager). Podaci se čitaju iz memorije za obradu, i upisuju na disk pomoću bafer menadžera. Kada sloj datoteka treba da obradi neku stranicu, on traži od bafer menadžera da preuzme stranicu, navodeći *rid*. Bafer menadžer preuzima stranicu sa diska, ako već nije u memoriji.

Prostorom na disku upravlja menadžer prostora na disku (eng. Disk Space Manager), u nastavku ćemo ga nazivati samo menadžer prostora. Kada sloju datoteka treba dodatni prostor za čuvanje novih slogova u datoteci, on traži od menadžera prostora da mu dodeli dodatnu stranicu za datoteku, takođe ga obaveštava kada mu više nije potrebna jedna od njegovih stranica na disku. Menadžer prostora vodi evidenciju o stranicama koje koristi sloj datoteka. Ako sloj datoteka oslobodi stranicu, menadžer prostora to evidentira, i kasnije ponovo koristi prostor ako sloj datoteka zatraži novu stranicu [1].

Graphical user interface, diagram, application

Description automatically generated

Slika 1.1. Arhitektura DBMS-a

U nastavku rada ćemo teorijski obraditi samu strukturu i organizaciju indeksa, nakon toga ćemo proći kroz sve indekse koje nudi Microsoft SQL Server i na praktičnim primerima pokazati način korišćenja i njihovu upotrebu.

# **Interna struktura i organizacija indeksa**

Najjednostavnija struktura fajla u bazi podataka je heap fajl. U heap fajlu, slogovi se čuvaju u nasumičnom redosledu na stranicama fajla. Ovakva organizacija fajla je pogodna kod povlačenja svih slogova ili pojedinačnog sloga čiji je identifikator zapisan u polju *rid*. Heap fajl ima jednostavnu strukturu i nije potrebna velika količina resursa za njegovo kreiranje. Međutim, ovakva organizacija fajla ne pruža lako i brzo pronalaženja podataka.

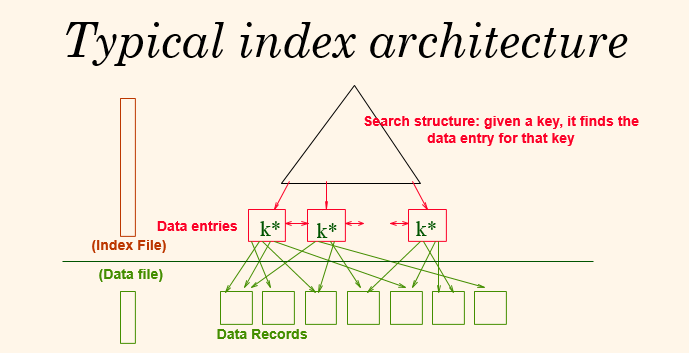
Indeksi su struktura podataka koja omogućava efikasno pretraživanje podataka na disku. Uz pomoć indeksa možemo brzo pronaći sve slogove koji zadovoljavaju zadate uslove pretrage u polju ključ traženja (eng. Search key) indeksa.

Indeksi omogućavaju brzo pronalaženje podataka tako što organizuju podatke na disku u strukturu stabla. Svaki čvor stabla sadrži vrednost ključa traženja i pokazivače na lokacije gde se nalaze slogovi na disku koji sadrže tu vrednost ključa. Kada se traži određeni slog, pretraga počinje od korena stabla i kreće se prema granama koje odgovaraju vrednostima ključa koji se traži. Na kraju pretrage, nalazimo lokaciju na disku gde se nalaze svi slogovi koji zadovoljavaju kriterijume pretrage [1].

Kada govorimo o indeksima, često se koristi termin „*data entry*“ za označavanje zapisa koji se čuvaju u datoteci indeksa. Svaki *data entry* u indeksu sadrži informaciju o jednom ili više redova podataka iz odgovarajuće tabele. Konkretno, svaki *data entry* ima jedinstvenu vrednost polja ključa traženja koje se koristi za indeksiranje.

Na primer, ako imamo tabelu korisnika u kojoj svaki red predstavlja jednog korisnika sa imenom, prezimenom i adresom. U indeksnoj datoteci za ovu tabelu, može se čuvati *data entry* za svaki red u tabeli koji sadrži vrednosti ključa traženja. *Data entry* sadrži informaciju o lokaciji reda u tabeli koji sadrži istu vrednost za polje ključa traženja (Slika 2.1).

Kada koristimo pretragu u indeksu, prvo pronalazimo odgovarajući *data entry* za traženu vrednost ključa traženja. Zatim, kada nađemo traženi zapis, koristimo informaciju o lokaciji reda u tabeli kako bismo došli do stvarnog reda podataka i prikupili sve informacije o korisniku (ime, prezime, adresa).



Slika 2.1 Tipična arhitektura indeksa

Dakle, indeksi nam omogućavaju efikasno pretraživanje podataka na osnovu vrednosti ključa traženja. Umesto da prolazimo kroz celu tabelu kako bismo našli željeni red, koristimo indekse da bismo pronašli slog koji se odnosi na taj red i tako brzo pristupili podacima.

## **Klasifikacija indeksa**

Kada je datoteka organizovana tako da je redosled slogova isti ili sličan redosledu *data entries* nekog indeksa, onda kažemo da je taj indeks **klasterovan** (eng. Clustered index), u suprotnom, indeks je **neklasterovan** (eng. Non-clustered index). Drugim rečima, indeks je klasterovan kada je datoteka fizički sortirana na osnovu vrednosti ključa traženja. Prednost klasterovanog indeksa je u tome što, kada se traži opseg vrednosti po ključu traženja, sistem može efikasno da preuzme sve slogove koji odgovaraju uslovu pretrage, bez pretraživanja cele datoteke.

Sa druge strane, kada je indeks neklasterovan, to znači da redosled slogova nije povezan sa redosledom *data entries*. Ovo može dovesti do problema sa performansama pri traženju jer će sistem možda morati da pretraži celu datoteku da bi pronašao odgovarajuće slogove. Na osnovu toga možemo zaključiti da cena korišćenja indeksa može značajno da varira u zavisnosti od toga da li je indeks klasterovan ili ne.

Kada se indeks kreira na skupu polja koja uključuju primarni ključ tabele, tj. ključ traženja sadrži primarni ključ relacije, tada se indeks naziva **primarni** (eng. Primary index). Ukoliko se indeks kreira na skupu polja koja ne uključuju primarni ključ, indeks se naziva **sekundarni** (eng. Secondary index). Sekundarni indeks može sadržati duplikate, jer *data entries* mogu da imaju iste vrednosti za polja ključ traženja. Međutim ako znamo da ključ traženja sadrži samo jedinstvene vrednosti, tj. ključ traženja sadrži neki od ključa kandidata, indeks nazivamo **jedinstvenim** (eng. Unique index) [1].

Diagram

Description automatically generated

Slika 2.2. Arhitektura klasterovanog i neklasterovanog indeksa

## **Struktura indeksa**

Heširanje je tehnika organizovanja slogova u indeksu koja nam omogućava brzo pronalaženje slogova koji imaju dati ključ traženja. Kod ove tehnike, slogovi se grupišu u *buckets*, gde jedan *bucket* sadrži primarnu stranicu i, po potrebi, dodatne stranice. Ukoliko želimo da pronađemo *bucket* kome pripada neki slog, to možemo odrediti primenom funkcije koja se zove heš funkcija [1].

Ideja heš funkcije je da za svaki ključ traženja izračuna jedinstvenu vrednost (heš vrednost) koja predstavlja lokaciju *bucket-a* u kojoj bi taj slog trebalo da se nalazi. Heš funkcija treba da bude dovoljno jednostavna i brza kako bi se što efikasnije izračunavala, ali istovremeno i dovoljno složena kako bi ravnomerno raspoređivala slogove po *bucket-ima* i smanjila mogućnost kolizije – situacije kada dva različita sloga imaju istu heš vrednost i samim tim bi bili smešteni u istu posudu.

Na *slici 2.3.* imamo primer dva heš indeksa, u prvom, podaci o zaposlenima se čuvaju u datoteci koja se hešira na osnovu *age*. Primenom heš funkcije na polje *age* identifikuje se stranica kojoj taj slog pripada. U ovom indeksu, *data entries* sadrži podatke *name*, *age* i *sal*. Heš funkcija u ovom primeru konvertuje vrednost ključa traženja u binarni zapis i koristi dva najniža bita kao identifikator *bucket-a*. U drugom indeksu koji za ključ traženja koristi *sal*, sadrži parove (*sal*, *rid*) kao *data entries*. Polje *rid* u ovom indeksu predstavlja pokazivač na slogove koji imaju vrednost *sal* za ključ traženja (na slici prikazano u obliku strelice).

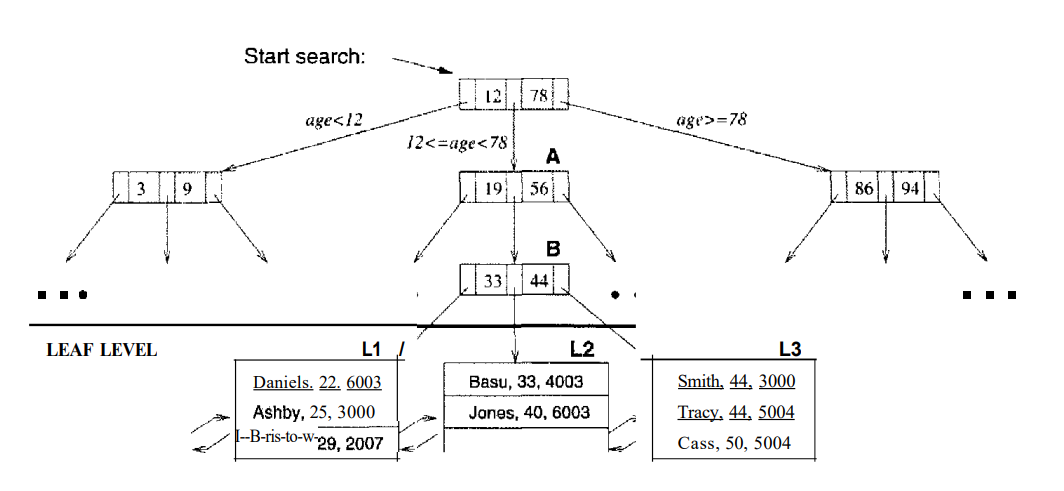
Diagram

Description automatically generated

Slika 2.3. Primer heš indeksa

Struktura indeksa zasnovana na stablu je popularnija u odnosu na strukturu koja je zasnovana na hešu. U strukturi zasnovanoj na stablu, *data entries* su organizovani u strukturi stabla i poređani u sortiranom redosledu po vrednosti ključa traženja. Pretraga za određenim slogom započinje u korenu stabla i prati putanju niz stablo, pri čemu svaki nivo „sužava“ prostor za pretragu dok se ne pronađe traženi slog. Stranice koje nisu listovi sadrže pokazivače čvorova zasnovane na ključu traženja, tj. pokazivač čvora koji ima ključ *k*, pokazuje na levo podstablo samo ukoliko su vrednosti ključa manje od *k*, ili pokazuje na desno podstablo samo ukoliko su vrednosti ključa veće ili jednake od *k*. Struktura stabla se održava umetanjem novih *data entries* u stablo kada se slogovi dodaju u datoteku, i brisanjem *data entries* kada se slogovi uklanjaju iz datoteke [1].

Na *slici 2.4.* imamo primer zaposlenih predstavljen strukturom indeksa zasnovanog na stablu. Ukoliko želimo da pronađemo sve zaposlene koji su u starosnoj granici između 24 i 50 godina, krećemo od korena stabla, ispitujemo uslov i usmeravamo se na čvor *A* koji zadovoljava uslov, zatim opet proveravamo uslov i usmeravamo se na čvor *B*. Zatim, ispitujemo sadržaj čvora *B* i dolazimo do lista *L1* koji sadrži tražene *data entries.* Ukoliko pogledamo listove *L2* i *L3*, vidimo da i oni takođe zadovoljavaju tražene uslove. Da bi se olakšalo pronalaženje takvih *data entries*, sve stranice su dvostruko ulančane, i sadrže pokazivače na prethodnu i sledeću stranicu (eng. next & prev pointer). Dakle, možemo preuzeti stranicu *L2*, koristeći pokazivač „*next*“ stranice *L1*, a zatim preuzeti stranicu *L3* koristeći pokazivač stranice *L2*.



Slika 2.4. Primer indeksa zasnovanog na stablu

Najčešći tip indeksne strukture zasnovane na stablu je B-stablo (eng. Balanced Tree).   
B+ stabla su samobalansirajuća stabla koja su optimizovana za pristup disku, ona eliminišu nedostatke B-stabla tako što čuvaju pokazivače samo na listovima. Ona su dizajnirana da minimizuju broj I/O operacija diska tako što obezbeđuju da su sve putanje od korena do lista iste dužine, odnosno da je struktura stabla uvek uravnotežena po visini. Broj I/O operacija je jednak visini stabla, tj. dužina puta od korena do lista plus broj listova stranica koji ispunjavaju uslove.

Ovo omogućava B+ stablima da podrže efikasne pretrage, dodavanja i brisanja čak i kada je datoteka sa podacima prevelika da bi stala u potpunosti u glavnu memoriju. Pronalaženje stranice je brže od binarne pretrage stranica u sortiranoj datoteci jer svaki čvor koji nije list može da primi veoma veliki broj pokazivača na čvorove, a visina stabla je retko veća od tri ili četiri u praksi.

U B+ stablu, svaki čvor koji nije list može imati različit broj dece. Prosečan broj dece za čvor koji nije list naziva se *fan-out* stabla. Ako svaki čvor koji nije list ima *n* dece, stablo visine *h* ima *nh* listova stranica. U praksi, čvorovi nemaju isti broj dece, ali koristeći prosečnu vrednost *F* za *n*, dobijamo da je broj listova stranica *Fh*. U praksi, *F* je najmanje 100, što znači da stablo visine četiri sadrži 100 miliona listova stranica. Dakle, možemo pretražiti datoteku sa 100 miliona listova stranica i pronaći stranicu koju želimo koristeći četiri I/O operacija. Nasuprot tome, za binarno traženje iste datoteke bi trebalo preko preko 25 I/O operacija.

Pored B i B+ stabla, postoje i druge strukture indeksa zasnovane na stablu, kao što su R stabla, koja su dizajnirana za indeksiranje višedimenzionalnih podataka kao što su prostorni podaci.

U opštem slučaju, indeksi omogućavaju efikasno pronalaženje podataka koji zadovoljavaju određeni uslov selekcije. Postoje dve vrste selekcije: selekcija po jednakosti i selekcija po opsegu. Selekcija po jednakosti je pretraga po traženoj vrednosti, npr. pretraga zaposlenih po datom ID-ju. Tehnike indeksiranja zasnovane na hešu su posebno efikasne za selekciju po jednakosti, jer izračunavanjem heš vrednosti ključa traženja, indeks može brzo da identifikuje *bucket* u kome se nalazi traženi slog. Međutim, indeksiranje zasnovano na hešu nije efikasno sa selekcijom po opsegu, jer slogovi koji se preuzimaju nemaju određenu vrednost, već spadaju u nekom opsegu vrednosti.

Selekcija po opsegu uključuje preuzimanje svih slogova koji spadaju u određeni opseg vrednosti, kao što smo imali na prethodnoj slici, pretraga zaposlenih u starosnom opsegu. Tehnike indeksiranja zasnovane na stablu, kao što je B+ stablo, posebno su efikasne za selekciju po opsegu, jer je stablo organizovano u sortiranom redosledu po ključu traženja. S obzirom da stablo ima uravnoteženu strukturu, može efikasno da locira tražene podatke sa malim broj I/O operacija, ono je efikasno i za selekciju po jednakosti.

U praksi, najbolja situacija je već imati sortirane fajlove, što se retko kad dešava. Međutim, i u takvim situacijama indeksi zasnovani na stablu imaju prednost nad sortiranim fajlovima, jer omogućavaju efikasno dodavanje i brisanje podataka, dok se kod sortiranih fajlova mora opet sortirati čitav fajl, što može biti veoma skupo. Druga stvar koju smo već spomenuli, pretraga po stablu je znatno brža nego binarna pretraga, koja najviše zavisi od veličine tog fajla i zahteva mnogo više I/O operacija [1].

# **Interna struktura i organizacija indeksa kod MS SQL Server-a**

Microsoft SQL Server predstavlja sistem za upravljanje realicionim bazama podataka (RDBMS). On je jedan od najpopularnijih i najčešće korišćenih DBMS-a na tržištu. Primarni jezik za upite je Transact SQL (T-SQL), koji proširuje SQL standarde uključujući lokalne promenljive, sačuvane procedure, trigere, funkcije, itd.

Stranica (eng. Page) je osnovna jedinica skladištenja podataka u SQL Serveru i veličina jedne stranice je 8KB. Skup od osam fizički susednih stranica predstavlja *extent* (veličine 64KB) i koristi se za efikasno upravljanje stranicama. Sve stranice su organizovane u *extent-ove*. Osim stranica podataka koje sadrže stvarne redove podataka koji su korisnici uneli, u SQL Serveru imamo i indeksne stranice koje sadrže reference o tome gde se podaci nalaze, kao i sistemske stranice koje čuvaju različite metapodatke o organizaciji podataka [2].

Svaka stranica u SQL Serveru počinje zaglavljem koje zauzima prvih 96B na stranici. Zaglavlje se koristi za skladištenje informacija kao što su broj stranice, vrsta stranice, količina slobodnog prostora na stranici. Zaglavlje stranice omogućava SQL Serveru da brzo i efikasno pristupa informacijama o svakoj stranici u bazi podataka, i da održava celokupnu strukturu baze podataka.

Svaki red podataka u SQL Serveru se serijski skladišti na stranici, odmah nakon zaglavlja. Nakon što su svi redovi podataka upisani na stranici, tabela pomeraja redova (eng. Row offset table) počinje odmah iza poslednjeg reda. Svaka tabela pomeraja sadrži po jedan unos za svaki red podataka na stranici. Svaki unos u tabeli pomeraja čuva informaciju o tome koliko je daleko prvi bajt reda od početka stranice. Uloga tabele pomeraja je da pomogne SQL Serveru da brzo locira redove na stranici. Unosi u tabeli pomeraja su u obrnutom redosledu od redosleda redova na stranici [2].

A picture containing chart

Description automatically generated

Slika 3.1. Struktura jedne stranice u SQL Serveru

PFS stranice (eng. Page Free Space) su vrsta stranica koje beleže status dodele svake stranice, da li je pojedinačna stranica dodeljena i koliko slobodnog prostora ima na svakoj stranici. PFS ima 1B za svaku stranicu, gde se beleži da li je stranica dodeljena, a ako jeste, da li je prazna, odnosno broj procenata koliko je popunjena. Ove informacije se koriste kada *DB Engine* mora da pronađe stranicu sa slobodnim prostorom gde će biti ubačen novi red. Indeksi ne zahtevaju da se prati slobodan prostor na stranici, jer se tačno zna gde treba dodati novi red na osnovu ključa indeksa.

Postoje dva osnovna tipa indeksa u SQL Serveru: klasterovani i neklasterovani indeksi. Obe vrste indeksa koriste već opisanu strukturu B+ stabla. Klasterovani indeks definiše fizičko uređenje podataka unutar tabele na osnovu ključa indeksa. Ukoliko tabela ima klasterovani indeks onda se ona naziva klasterovana tabela, a ukoliko nema, podaci se čuvaju u neuređenoj strukturi i tabela se naziva *heap*. Tabela može imati samo jedan klasterovani indeks, ali može imati više neklasterovanih indeksa. Razlika je u tome što se u klasterovanom indeksu listovi sastoje od slogova (svih kolona tabele), dok se u neklasterovanom indeksu listovi sastoje od pokazivača – *rid* (identifikator sloga na osnovu koga možemo odrediti stranicu koja ga sadrži) [3].

Pored klasterovanih i neklasterovanih indeksa, SQL Server podržava i druge vrste indeksa, kao što su Full-Text, XML i prostorni indeksi (eng. Spatial Indexes). Indekse možemo klasifikovati i na osnovu formata skladištenja, kao što je prikazano u *tabeli 1*.

Tabela 1. Klasifikacija indeksa na osnovu formata skladištenja

|  |  |
| --- | --- |
| Format skladištenja | Tip indeksa |
| Disk-based rowstore | Clustered |
| Nonclustered |
| Unique |
| Filtered |
| Columnstore | Clustered columnstore |
| Nonclustered columnstore |
| Memory-optimized | Hash |
| Memory-optimized nonclustered |

U nastavku poglavlja ćemo se detaljnije baviti sa svakim od ovih pojedinačno, prolazeći kroz njihovu strukturu i način upotrebe u SQL Serveru.

## **Klasterovani indeksi**

Klasterovani indeks određuje fizički redosled podataka u tabeli na osnovu vrednosti u jednoj ili više kolona. To znači da su redovi podataka u tabeli sa klasterovanim indeksom fizički sortirani i čuvaju se u redosledu vrednosti ključa klasterovanog indeksa. Pošto može postojati samo jedan klasterovani indeks po tabeli, važno je odabrati pravu kolonu ili kolone za kreiranje klasterovanog indeksa. Kolone treba da ispune sledeće kriterijume:

* Često se koriste u upitima – indeks treba definisati nad kolonom ili skupu kolona koje se često koriste u upitima, kako bi se poboljšale performanse upita
* Imaju visok stepen jedinstvenosti – kolone ili skup kolona trebaju biti jedinstvene, jer će indeks po tim kolonama vršiti sortiranje podataka i ne bi trebalo biti duplikata
* Podržavaju upite opsega – kolone ili skup kolona trebaju da podržavaju upite opsega, tj. upite koje koriste operatore poređenja kako bi se dobio opseg vrednosti

Napomena: Kada kreiramo ograničenje *PRIMARY KEY*, automatski se kreira jedinstveni indeks (eng. Unique index), koji je podrazumevano klasterovani.

Svaka stranica u indeksnom B+ stablu se naziva indeksni čvor. Početni čvor, tj. čvor u najvišem nivou se naziva koren, dok se čvorovi u najnižem nivou stabla nazivaju listovi. Svi nivoi između korena i listova se nazivaju središnji nivoi. Kod klasterovanog indeksa, listovi sadrže stranice podataka tabele, dok koren i čvorovi središnjeg nivoa sadrže indeksne stranice, a one sadrže indeksne redove. Svaki indeksni red sadrži ključ i pokazivač na stranicu u središnjem nivou ili na red podataka u listovima indeksa. Da bi se omogućilo brže pretraživanje i sortiranje podataka, stranice u svakom nivou indeksa su povezane u dvostruko lančanu listu. Ovo znači da svaka stranica sadrži pokazivač na prethodnu i sledeću stranicu u istom nivou indeksa [4].

Diagram

Description automatically generatedKlasterovani indeksi imaju po jedan red u *sys.partitions*, sa vrednošću *index\_id = 1*, za svaku particiju koju koristi indeks. Podrazumevano, klasterovani indeks ima samo jednu particiju. Međutim, kada klasterovani indeks ima više particija, svaka particija ima strukturu B+ stabla koja sadrži podatke za tu određenu particiju. Na primer, ako klasterovani indeks ima četiri particije, postoji četiri strukture B+ stabla.

Kada se kreira klasterovani indeks sa više particija, SQL Server automatski kreira particije na osnovu neke od ključnih vrednosti koje se koriste u indeksu. SQL Server koristi algoritme za raspodelu podataka kako bi se omogućilo ravnomerno raspoređivanje podataka po particijama. Na *slici 3.2.* je prikazana struktura klasterovanog indeksa u jednoj particiji [4].

Slika 3.2. Struktura klasterovanog indeksa

U nastavku ćemo pokazati primer korišćenja klasterovanog indeksa u SQL Serveru. Najpre ćemo kreirati tabelu Employee sa odgovarajućim kolonama (*Primer 3.1.*) i dodati 100.000 redova. Kao što smo napomenuli prilikom dodavanja ograničenja za primarni ključ, automatski će se kreirati i klasterovani indeks nad primarnim ključem. Ukoliko želimo da sami kreiramo klasterovani indeks, to možemo učiniti T-SQL naredbom na *primeru 3.2.*

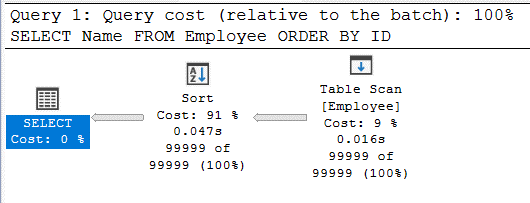


Primer 3.1. Naredba za kreiranje tabele Employee



Primer 3.2. Naredba kreiranja klasterovanog indeksa

Na levoj slici je prikazan plan izvršenja upita bez indeksa, dok je na desnoj nakon dodavanja indeksa, i kao što možemo videti dodavanjem indeksa smo „uklonili“ operaciju sortiranja, koja je jako skupa.

Text, letter

Description automatically generated

Slika 3.3. Plan izvršenja upita pre i nakon dodavanja klasterovanog indeksa

## **Neklasterovani indeksi**

Neklasterovani indeks je vrsta indeksa koja sadrži vrednost ključa i pokazivač na redove gde se podaci nalaze – *rid*. Uglavnom se koriste za poboljšanje performansi upita koji se često koriste, a nisu obuhvaćeni klasterovanim indeksom.

Kao i klasterovani indeksi imaju istu strukturu B+ stabla, sa malim razlikama:

* Redovi podataka nisu sortirani i skladišteni na osnovu ključa indeksa
* Listovi u stablu se sastoje od indeksnih redova, umesto stranica podataka, kao kod klasterovanih. Svaki indeksni red sadrži ključne kolone, uključene kolone (eng. included column) i pokazivače na osnovu koga se može locirati stranica koja ga sadrži

Kada kreiramo neklasterovani indeks, ukoliko se radi o tabeli koji nema klasterovani indeks, tj. ukoliko je tabela *heap*, onda će pokazivač kod neklasterovanog indeksa biti pokazivač na red, poznatiji je kao *rid* (u formatu: *id\_fajla:stranica:slot*). Ukoliko tabela ima klasterovani indeks, pokazivač će biti vrednost ključ klasterovanog indeksa za taj red [4].

Diagram

Description automatically generatedNeklasterovani indeksi imaju po jedan red u *sys.partitions*, sa vrednošću *index\_id > 1*, za svaku particiju koju koristi indeks. Podrazumevano, neklasterovani indeks ima samo jednu particiju. Ukoliko ima više particija, ista je situacija kao kod klasterovanih indeksa, svaka particija ima strukturu B+ stabla koja sadrži podatke za tu određenu particiju.

Kada indeks sadrži sve kolone koje je koriste u nekom upitu, onda se može reći da on „pokriva“ upit. To znači da su sve potrebne informacije prisutne u indeksu, bez potrebe za pristupom tabeli ili klasterovanom indeksu. Kada se indeks poklapa sa upitom, može se smanjiti broj čitanja sa diska i poboljšati performanse upita. Ovu funkcionalnost možemo postići uključivanjem kolona kod neklasterovanih indeksa. Ove kolone se smeštaju samo u nivou lista i ne utiču na broj kolona ključa indeksa ili veličinu ključa indeksa [4].

Slika 3.4. Struktura neklasterovanog indeksa

T-SQL naredba kreiranja neklasterovanog indeksa nad kolona *Salary* i *Age* tabele *Employee* je prikaza u *primeru 3.3.*. Zatim ćemo uporediti performanse upita (*Primer 3.4*.), sa i bez neklasterovanog indeksa.



Primer 3.3. Naredba kreiranja neklasterovanog indeksa



Primer 3.4. Upit opsega nad tabelom Employee

Text

Description automatically generatedGraphical user interface, text, application

Description automatically generated

Slika 3.5. Plan izvršenja upita pre i nakon dodavanja neklasterovanog indeksa

Na levoj slici imamo plan izvršenja upita pre dodavanja indeksa, možemo videti da SQL Server koristi operaciju *Clustered* *Index Scan* da pristupi podacima, što znači da se čitaju svi redovi u indeksu kako bi se pronašli traženi. Nakon dodavanja indeksa, na desnoj slici, možemo videti da SQL Server koristi operaciju *Index Seek* da pristupi podacima, što znači da se pristupa samo onim redovima koji su potrebni za izvršavanje upita. Kao što možemo videti efikasnija operacija je *Index Seek*, pa možemo reći da smo dodavanjem neklasterovanog indeksa poboljšali performanse.

Sada možemo preći na primer kreiranja neklasterovanog indeksa sa uključenim kolonama (*Primer 3.5.*). Dakle, ovakav indeks nam može biti korisan ako želimo vratiti neke kolone, koje ne želimo da stavimo u ključ indeksa, ili jednostavno indeks ne podržava taj tip kolone za ključ. Na *primeru 3.6* imamo upit koji vraća informacije o zaposlenima koji imaju između 18 i 25 godina. Kao ključ indeksa ćemo postaviti *Age*, dok ćemo ostale kolone dodati u opciju *INCLUDE*.



Primer 3.5. Kreiranje neklasterovanog indeksa sa uključenim kolonama



Primer 3.6. Upit opsega na kolonom Age

Text

Description automatically generatedText

Description automatically generated

Slika 3.6. Plan izvršenja upita pre i nakon dodavanja neklasterovanog indeksa sa uključenim kolonama

## **Jedinstveni indeksi**

Jedinstveni indeksi (eng. Unique indexes) obezbeđuju da su vrednosti ključeva indeksa jedinstvene, što znači da nijedan red u indeksiranoj tabeli nema iste vrednosti za indeksirane kolone. Ovaj tip indeksa se može kreirati na jednoj ili više kolona.

Jedinstveni indeksi su korisni u situacijama kada sami podaci zahtevaju jedinstvenost, na primer kada se radi o identifikatorima koji moraju biti jedinstveni u svim zapisima. Kreiranjem jedinstvenih indeksa osiguravamo integritet podataka, tj. baza podataka će primeniti ograničenje jedinstvenosti i sprečiti upisivanje duplikata u tabelu.

I klasterovani i neklasterovani indeksi mogu biti jedinstveni. Kreiranjem ograničenja *PRIMARY* *KEY* ili *UNIQUE*, automatski će se kreirati i jedinstveni indeks za navedene kolone. Ukoliko se kreira klasterovani ili neklasterovani indeks nad kolonom ili skupom kolona koje su jedinstvene, uvek je preporuka kreirati jedinstveni indeks, jer se tako pružaju dodatne informacije optimizatoru upita (eng. Query Optimizator) koji kasnije može generisati efikasnije planove izvršenja [4].

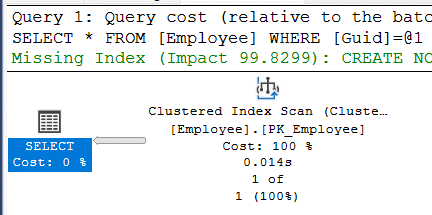
Na *primeru 3.7*. imamo datu naredbu kreiranja jedinstvenog neklasterovanog indeksa nad kolonom *Guid*. Nakon toga ćemo izvršiti pretragu po ovoj koloni i uporediti planove izvršenja.



Primer 3.7. Kreiranje jedinstvenog indeksa nad kolonom Guid



Primer 3.8. Pretraga zaposlenih po Guid-u

Diagram, schematic

Description automatically generated

Slika 3.7. Plan izvršenja upita pre i nakon dodavanja jedinstvenog indeksa

U planu izvršenja pre dodavanja indeksa, SQL Server je iskoristio operaciju *Clustered Scan Index*, tj. izvršio je pretraživanje podataka u klasterovanom indeksu, što u ovom slučaju nije optimalno. Takođe, SQL Server nam je predložio kreiranje neklasterovanog indeksa. Nakon kreiranja jedinstvenog indeksa nad kolonom *Guid*, vidimo da je SQL Server iskoristio operaciju *Index Seek*, tj. pretraživao je redove po ključu indeksa, a zatim je izvršio operaciju *Key Lookup* nad klasterovanim indeksom, jer smo tražili sve kolone kao povratne (*SELECT \* ...*). Ovu operaciju možemo izbeći ukoliko dodamo sve povratne kolone kao uključene (*INCLUDE*) i tako kreiramo pokrivajući indeks, o kome je bilo više reči u prethodnom poglavlju.

## **Filtrirani indeksi**

Filtrirani indeks (eng. Filtered index) je tip neklasterovanog indeksa koji se kreira sa *WHERE* klauzulom, koja specificira predikat filtera koji ograničava indeks na određeni podskup redova u tabeli. Ovaj tip indeksa je posebno koristan kada velika tabela sadrži mnogo redova, a upiti često pristupaju samo određenom podskupu redova [4].

Prednosti filtriranog indeksa:

* Poboljšanje performansi upita i kvaliteta plana izvršenja: Filtrirani indeksi su manji od neklasteriranih indeksa koji pokrivaju celu tabelu i imaju filtriranu statistiku. Filtrirana statistika je preciznija od statistike koja pokriva celu tabelu, jer obuhvata samo redove u filtriranom indeksu.
* Smanjenje troškova održavanja indeksa: Filtrirani indeks se održava samo kada se podaci u indeksu menjaju pomoću DML naredbi. Filtrirani indeks smanjuje troškove održavanja indeksa u poređenju sa indeksom koji pokriva celu tabelu, jer je manji i održava se samo kada su podaci u indeksu promenjeni. Moguće je imati veliki broj filtriranih indeksa, posebno kada sadrže podatke koji se retko menjaju.
* Smanjenje troškova skladištenja indeksa: S obzirom da se filtrirani indeks kreira na određenom podskupu podataka, samim tim je i veličina indeksa manja, što znači manje zauzetog prostora na disku. Ovo može biti korisno u situacijama kada je potrebno smanjiti veličinu baze podataka i troškove skladištenja, a i olakšava održavanje baze podataka jer se manji indeksi brže ažuriraju.

Filtrirani indeksi su posebno korisni kada tabela ima kolone koje sadrže dobro definisane podskupove podataka koji se često koriste u *SELECT* naredbama. Primeri takvih kolona su kolone koje sadrže ogroman broj *NULL* vrednosti, heterogene kolone koje sadrže kategorije podataka, kolone koje sadrže opsege vrednosti kao što su iznosi, vreme i datumi.

U našem primeru kolona *EndDate* koja predstavlja datum kada je radnik prestao da radi u kompaniji je dobar kandidat za ključ filtriranog indeksa, jer postoji dosta redova koji imaju *NULL* vrednost. Najpre ćemo kreirati filtrirani neklasterovani indeks u kome će predikat biti da *EndDate* nema *NULL* vrednost (*Primer 3.9*). Nakon toga ćemo pretražiti sve zaposlene koji više nisu zaposleni u kompaniji (*Primer 3.10*) i uporediti planove izvršenja indeksa.



Primer 3.9. Kreiranje filtriranog indeksa



Primer 3.10. Pretraga svih zaposlenih koji više ne rade u kompaniji

Text

Description automatically generatedText

Description automatically generated

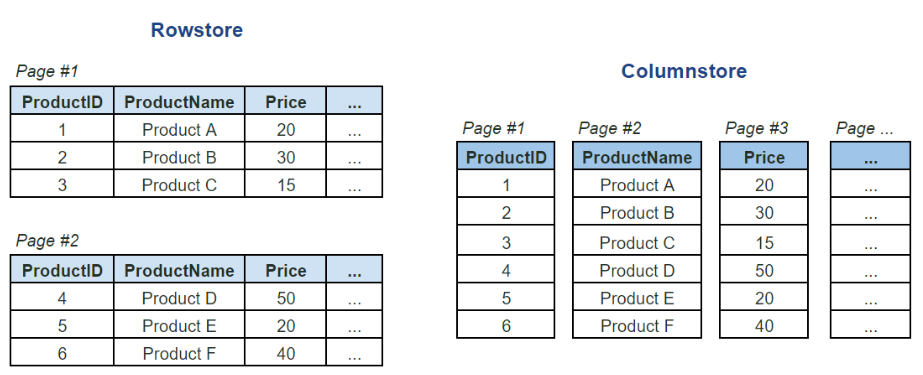
Slika 3.8. Plan izvršenja upita pre i nakon dodavanja filtriranog indeksa

Kao što možemo videti, i pre i nakon dodavanja indeksa SQL je iskoristio operaciju *Index Scan*, s tim što je nakon dodavanja indeksa SQL Server izvršio pretragu u dodatom filtriranom indeksu, što se odrazilo na brže vreme izvršenja (*Slika 3.8*).

## **Columnstore indeksi**

*Rowstore* i *columnstore* su dva različita formata skladištenja u sistemima za upravljanje relacionim bazama podataka. *Rowstore* je tradicionalni način čuvanja podataka, gde se podaci čuvaju u formatu red po red. Kod *rowstore* formata, sve kolone za jedan red se čuvaju zajedno na stranici. To znači da ako treba da preuzmemo samo određene podatke, moramo pročitati ceo red, čak iako su nam potrebne samo nekoliko kolona. Ovaj format je optimizovan za efikasno preuzimanje jednog reda ili malog opsega redova [5].

Sa druge strane, kod *columnstore* formata, podaci se čuvaju u formatu kolona po kolonu. Kod *columnstore* formata, sve vrednosti za jednu kolonu se čuvaju zajedno na stranici. Ovaj format je optimizovan za efikasno pronalaženje podataka za određenu kolonu.

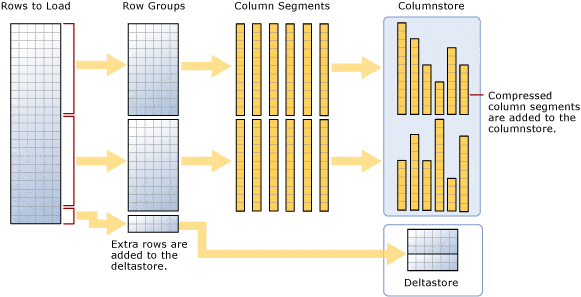


Slika 3.9. Skladištenje podataka u rowstrore i columnstore formatu

Columnstore indeks je tip indeksa koji većinu podataka čuva u *columnstore* formatu. To znači da se podaci u ovom formatu kompresuju i dekompresuju po kolonama. Kada se izvršava upit koji zahteva pristup samo određenoj koloni, nije potrebno dekompresovati sve vrednosti u svakom redu kako bi se dobila tražena vrednost. Ovo omogućava brzo pretraživanje cele kolone u velikoj tabeli.

Columnstore indeksi takođe fizički skladište neke podatke u *rowstore* formatu koji se naziva *deltastore*. *Deltastore* je mesto za skladištenje redova koji su premali da bi se kompresovali u *columnstore* formatu. Svaki *deltastore* se sastoji iz više *delta* *rowgroup* i svaki je implementiran kao klasterovano B+ stablo indeksa i koristi se za skladištenje redova dok ne dostignu određeni limit, nakon čega se od njih kreira nova kompresovana grupa.

Columnstore indeks u SQL Serveru funkcionišu tako što prvo dele tabelu u *rowgroups*, koje sadrže najviše 1.048.576 redova. Neophodno je da budu dovoljno velike da podrže stepen kompresije koji ova tehnika pruža. Kada je *rowgroup* pun, zatvara se i prebacuje u *columnstore* format. Svaka grupa sadrži po jedan *column segment* za svaku kolonu u tabeli, i svaki od njih je kompresovan i smešten na disku (*Slika 3.10*). S obzirom da su svi podaci unutar kolona iz istog domena i da je moguće često ponavljanje vrednosti, ova tehnika pruža neverovatnu kompresiju, koja često može da smanji zauzeće na disku [5].



Slika 3.10. Učitavanje podataka za columnstore indeks

Columnstore indeksi su optimizovani za radna opterećenja skladišta podataka (eng. Data Warehouse) koja uglavnom obavljaju masovna učitavanja i složene upite nad ogromnom količinom podataka. Oni takođe nude visoku stopu kompresije i mogućnost paralelne obrade upita.

Columnstore indeksi mogu biti i klasterovani i neklasterovani. Počev od SQL Servera 2016 moguće je kombinovati rowstore i columnstore indekse nad jednom tabelom. U našem primeru zaposlenih kreiraćemo neklasterovani columnstore indeks nad kolonom *Age*, a zatim ćemo grupisati zaposlene po godinama, kako bi videli njihovu prosečnu platu (*Primeri 3.11* i *3.12*).



Primer 3.11. Kreiranje neklasterovanog columnstore indeksa



Primer 3.12. Upit koji grupiše zaposlene po godinama i vraća njihovu prosečnu platu

A picture containing text

Description automatically generatedGraphical user interface

Description automatically generated with medium confidence

Slika 3.11. Prikaz I/O statistike i plana izvršenja upita pre dodavanja columnstore indeksa



Graphical user interface

Description automatically generated

Slika 3.12. Prikaz I/O statistike i plana izvršenja nakon dodavanja columnstore indeksa

Pre dodavanja indeksa broj logičkih čitanja je bio 3475, dok se nakon kreiranja columnstore indeksa taj broj smanjio na 0. Ukupno vreme izvršenja upita se skoro prepolovilo, a vidimo da se i vreme CPU smanjilo na 0. Pa možemo zaključiti da smo postigli ogroman dobitak na performansama (*Slike 3.11* i *3.12*).

## **Memorijski optimizovani indeksi**

Memorijski optimizovane tabele (eng. Memory-optimized table) su tip tabela u SQL Serveru koje su dizajnirane da obezbede brži pristup podacima čuvanjem cele tabele u memoriji. Ključna karakteristika je da su one potpuno trajne, tj. sve promene podataka u tabeli se zadržavaju na disku kako bi se obezbedila trajnost podataka. Dakle, primarno skladište za memorijski optimizovane tabele je glavna memorija, dok se kopija podataka čuva na disku. Transakcije nad memorijski optimizovanim tabelama takođe u potpunosti zadovoljavaju ACID svojstva [6].

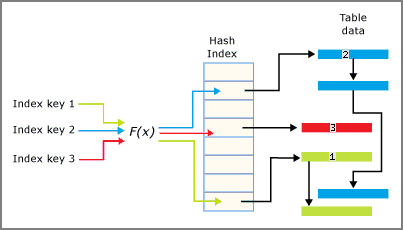
Prilikom kreiranja tabele neophodno je kreirati i bar jedan indeks, a indeksi definisani nad ovim tabelama se zovu memorijski optimizovani indeksi (eng. Memory-Optimized indexes). Ova vrsta indeksa se razlikuje od tradicionalnih indeksa koji se čuvaju na disku, a neke od glavnih razlika su [6]:

* Redovi podataka se ne čuvaju na indeksnim stranicama, već u memoriji, tako da ne postoji kolekcija stranica, tj. *extents*
* Promene napravljene u ovim indeksima se ne zadržavaju odmah na disku, sve dok se transakcija ne završi. Kada transakcija modifikuje podatke u memorijski optimizovanoj tabeli, promene podataka se upisuju u dnevnik transakcija, koji služi za održavanje trajnosti i konzistentnosti podataka
* Ukoliko se server isključi ili sruši, svi podaci sačuvani u memoriji se gube, tako da se memorijski optimizovani indeksi ponovo „grade“ kada baza postane dostupna

Postoje 2 vrste memorijski optimizovanih indeksa:

1. Heš indeksi
2. Neklasterovani indeksi

Heš indeksi su jedina vrsta indeksa u SQL Serveru koja koristi heš indeksnu strukturu, objašnjenu u poglavlju *2.2*. Heš indeks se sastoji od niza pokazivača, a svaki element se naziva heš *bucket*. Svaki heš *bucket* je veličine 8B. Primenom heš funkcije na kolone indeksnog ključa se određuje u koji *bucket* će slog biti smešten. Svi slogovi koji imaju istu vrednost heš funkcije se smeštaju u isti *bucket* i povezani su lancem pokazivača na svoju lokaciju u memorijski optimizovanoj tabeli (*Slika 3.13*).



Slika 3.13. Pretraga podataka pomoću heš indeksa

*DB Engine* ima jednu heš funkciju koja se koristi za sve heš indekse. Heš funkcija je deterministička, tj. ista vrednost ulaznog ključa će uvek biti mapirana u isti *bucket*. Ona je i balansirana, tj. prati Poasonovu ili normalnu raspodelu, a ne ravnomernu linearnu raspodelu. Ako se dva ključa indeksa mapiraju na isti heš *bucket*, dolazi do kolizije. Veliki broj kolizija može imati negativan uticaj na performanse. Realistično je da oko 30% *bucket-a* sadrži duplikate [6].

Broj *bucket-a* u heš indeksu idealno bi trebao da bude 1-2 puta veći od broja različitih vrednosti u ključu indeksa. Međutim, često nije moguće unapred predvideti koliko će vrednosti ključa biti prisutno u indeksu. U tim slučajevima, dobro je postaviti broj *bucket-a* na vrednost koja je do 10 puta veća od stvarnog broja vrednosti ključa. Ako je broj *bucket-a* premali, dolazi do povećanja broja kolizija i usporavanja pretrage indeksa, jer se ključevi sa istom heš vrednošću moraju čuvati u istom *bucket-u*, pa može doći do preopterećenja i sporog pristupa tim ključevima. Sa druge strane, ako broj *bucket-a* nije dovoljno velik, postoji rizik od gubitka memorije i prevelikog troška memorije za indeks.

Memorijski optimizovani neklasterovani indeksi koriste strukturu Bw-stabla, koje je razvio Microsoft 2011. godine. Bw-stablo je optimizovana verzija B-stabla koja je efikasnija za korišćenje podataka koji se nalaze u memoriji. Bw-stablo se sastoji od 3 glavne komponente: *PidMap*, *PidAlloc* i skup povezanih stranica. *PidMap* je mapa stranica organizovanih po ID-ju, *PidAlloc* se koristi za dodeljivanje i ponovno korišćenje ID-jeva stranica, a skup povezanih stranica čini unutrašnju strukturu Bw-stabla.

Stranice indeksa sadrže skup uređenih vrednosti podataka po ključu indeksa. Međutim, ove stranice nemaju fiksnu veličinu i ne mogu da se menjaju nakon što su kreirane. Slogovi u nivou lista sadrže pokazivače na lanac slogova koji imaju istu vrednost ključa indeksa, slično kao kod heš indeksa gde su više slogova uzastopno povezani. Koren i međunivoi se nazivaju internim stranicama i kao kod B-stabla upućuju na naredni, dublji, nivo indeksa, sa tim da umesto da ukazuju na samu stranicu oni koriste logički identifikator stranice, koji predstavlja poziciju u posebnoj strukturi nalik nizu koja se zove *mapping table*. Svaki element ove strukture sadrži pokazivač na stvarnu indeksnu stranicu [6].

Ako stranica na nivou lista mora da se promeni, *DB Engine* ne menja samu stranicu. Umesto toga, on kreira *delta* *slog* koji opisuje promenu i dodaje ga na prethodnu stranicu. Zatim takođe ažurira adresu *mapping table* za prethodnu stranicu, da pokazuje na adresu *delta* sloga koji sada postaje fizička adresa za ovu stranicu (*Slika 3.14*).

Diagram

Description automatically generated

Slika 3.14. Dodavanje delta slogova

U sledećem primeru ćemo kreirati memorijski optimizovanu tabelu, i prilikom kreiranja ćemo definisati jedan memorijski optimizovan heš indeks nad kolonom *Name* i jedan neklasterovani indeks nad primarnim ključem tabele.



Primer 3.13. Kreiranje memorijski optimizovane tabele i definisanje neklasterovanog i heš indeksa

## **XML indeksi**

XML indeksi se mogu kreirati na kolonama u tabeli ukoliko kolona ima tip podataka XML. Ovi indeksi pomažu u poboljšanju performansi upita indeksirajući sve tagove, vrednosti i putanje unutar smeštenih XML instanci.

U SQL Serveru, XML instance se čuvaju u kolonama tipa XML kao veliki binarni objekti (*BLOB*). Kada se upit izvrši na koloni tipa XML, baza podataka treba da odseče *BLOB* tokom izvršenja, tj. da pretvori XML format u relacioni format kako bi mogao biti procesiran od strane SQL-a. Proces odsecanja može biti dugotrajan, pogotovo za ogromne XML podatke. Međutim, kreiranjem XML indeksa, baza podataka može indeksirati sve tagove, vrednosti i putanje unutar XML instanci. To znači da kada izvršimo upit, baza podataka može da koristi XML indeks da brzo locira relevantne informacije bez potrebe da odseče ceo *BLOB* tokom izvršenja. Kao rezultat toga, vreme obrade upita može biti značajno smanjeno [8].

XML indeksi su u SQL Serveru podeljeni u dve kategorije: primarni i sekundarni XML indeks. Primarni XML indeks je interna struktura koju kreira SQL Server za skladištenje XML podataka radi efikasne obrade upita. Automatski se kreira kada je XML kolona definisana u tabeli i ne može se obrisati ili izmeniti. Sekundarni XML indeksi su korisnički definisani indeksi i mogu se kreirati samo ukoliko već postoji primarni indeks. SQL Server podržava tri tipa sekundarnih XML indeksa: *PATH*, *VALUE* i *PROPERTY* [8].

T-SQL naredbe kreiranja primarnog i *PATH* sekundarnog indeksa su date u *primeru 3.14*, za kreiranje *VALUE* ili *PROPERTY* sekundarnih indeksa samo je potrebno navesti *VALUE* ili *PROPERTY* umesto *PATH*.



Primer 3.14. Kreiranje primarnog i sekundarnog (PATH) XML indeksa

*PATH* indeksi čuvaju skup putanja koje odgovaraju XML elementima. Putanje se često koriste za lociranje određenih čvorova unutar XML instance tako što se navede putanja do čvora na osnovu njegove hijerarhije. Ukoliko upit uključuje izraze putanja ili vrednosti čvorova, SQL Server može da iskoristi *PATH* indeks da brzo locira redove u primarnom indeksu koji odgovara navedenoj putanji.



Primer 3.15. Upit zasnovan na izrazima putanje korišćenjem exist metode

U upitu u *primeru 3.15*, izraz putanje */PD:ProductDescription/@ProductModelID* i vrednost *"19"* odgovaraju ključnim poljima *PATH* indeksa. Ovo omogućava direktno traženje u *PATH* indeksu i obezbeđuje bolje performanse pretrage.

Ukoliko su upiti zasnovani na vrednosti, kao u *primeru 3.16*, a putanja nije u potpunosti navedena, efikasnije rezultate pretrage možemo dobiti kreiranjem *VALUE* sekundarnog indeksa. U ovom primeru broj telefona (*number*) nam je poznat, ali nismo sigurni tačno gde i koliko puta se može pojaviti u XML instanci kao dete čvora *telephoneNumber*.



Primer 3.16. Upit zasnovan na vrednosti korišćenjem exist metode

*PROPERTY* indeks je izgrađen na svojstvima XML instanci. Kada upit uključuje *value()* metodu u kojoj se navode imena svojstava, SQL Server može da koristi *PROPERTY* indeks kako bi brže locirao redove koji odgovaraju navedenom svojstvu u primarnom indeksu.

Na primer, ukoliko imamo tabelu koja u koloni XML čuva informacije o proizvodu. Ukoliko želimo da pronađemo cenu određenog proizvoda sa poznatom vrednošću primarnog ključa, možemo iskoristiti value metodu, kao u sledećem primeru.



Primer 3.17. Upit zasnovan na vrednosti korišćenjem value metode i poznatog ID proizvoda

Izbor tipa sekundarnog XML indeksa najviše zavisi od tipova upita koji se izvršavaju nad XML podacima. Kreiranjem odgovarajućeg indeksa možemo poboljšati performanse upita i smanjiti vreme izvršenja.

## **Prostorni indeksi**

SQL Server pruža podršku za prostorne podatke (eng. Spatial data) i prostorne indekse (eng. Spatial indexes). Prostorni podaci su georeferencirani, tj. svaki podatak je povezan sa stvarnom lokacijom na površini zemlje. Prostorni indeks je vrsta indeksa koji omogućava da se indeksiraju prostorne kolone. Prostorne kolone su kolone tabele koje sadrže tipove prostornih podataka, kao što su *geometry* i *geography* [7].

U SQL Serveru, prostorni indeksi koriste strukturu B-stabla, zbog efikasnije organizacije i pretrage velikih količina podataka. Da bi se kreirao prostorni indeks, 2D prostorni podaci se moraju predstaviti u linearnom obliku kako bi bili skladišteni u B-stablu. Oni funkcionišu tako što dele prostorne podatke u skup manjih delova – ćelija, a zatim kreiraju indeks za svaku ćeliju.

Prostorni indeksi pomažu da se ubrzaju prostorni upiti, tj. upiti nad prostornim podacima. Kod prostornih upita, kao i kod „običnih“, takođe možemo vršiti spojeve – ovde se nazivaju prostorni spojevi (eng. Spatial join). Prostorni spoj povezuje dve tabele na osnovu prostornih odnosa njihovih prostornih atributa. Prostorni spoj kombinuje dva ili više skupa prostornih podataka u odnosu na uslov upita i prostorne operacije. SQL Server podržava sledeće prostorne operacije: *STContains()*, *STDistance()*, *STEquals()*, *STIntersects()*, *STOverlaps()*, *STTouches()*, i *STWithin()* [7].

U našem primeru ćemo kreirati tabelu sa prostornom kolonom (*the\_geom*), zatim dodati prostorni indeks nad istom kolonom (*Primer 3.18*), i izvršiti upit pretrage koji pronalazi sve tačke koje se nalaze na rastojanju manjem od 1km od zadatke tačke (*POINT(43.3267515, 21.8933164)*) (*Primer 3.19*).



Primer 3.18. Kreiranje tabele sa prostornom kolonom the\_geom i dodavanje indeksa nad istom



Primer 3.19. Prostorni upit koji pretražuje sve tačke koje su udaljene manje od 1km od zadate tačke

## **Full-text indeksi**

*Full-text* indeksi su vrsta indeksa u SQL Serveru koja omogućava brzo i efikasno pretraživanje podataka na osnovu karaktera. Oni se mogu kreirati na jednoj ili više kolona čiji je tip podataka zasnovan na karakterima, kao što su: *char*, *varchar*, *nchar*, *nvarchar*, *text*, *ntext*, *image*, *xml*, *varbinary* i *FILESTREAM* [9].

Svaki *full-text* indeks može indeksirati jednu ili više kolona u tabeli, a svaka kolona može koristiti poseban jezik. Ovo je bitno jer različiti jezici imaju različita pravila za razdvajanje reči, što se koristi za određivanje osnovne forme reči. Na primer, srpski jezik ima različita pravila za razdvajanje reči od engleskog jezika, pa *full-text* indeks koji je kreiran na „srpskoj“ koloni možda neće efikasno raditi na „engleskoj“ koloni.

Prilikom kreiranja *full-text* indeksa, SQL Server kreira invertovani indeks koji mapira svaku reč u indeksiranim kolonama na redove koji sadrže tu reč. To omogućava brzo i efikasno pretraživanje indeksiranih kolona koristeći predikate kao što su *CONTAINS*, *FREETEXT* ili *FREETEXTTABLE*.

*Full-text* indeksi mogu zauzeti značajnu količinu prostora na disku, posebno za velike tabele sa mnogo indeksiranih kolona. Osim toga, *full-text* indeksi mogu uticati na performanse CRUD operacija u tabeli, jer svaka modifikacija u indeksiranim kolonama mora takođe da ažurira full-text indeks [9].

Pre nego što kreiramo *full-text* indeks, potrebno je kreirati katalog (eng. Full-Text Catalog). U ovom katalogu se čuvaju podaci o *full-text* indeksu. Jedan katalog može da sadrži više indeksa, ali *full-text* indeks može biti samo u jednom katalogu. Takođe, nad svakom tabelom možemo kreirati samo jedan *full-text* indeks (*Primer 3.20*).



Primer 3.20. Kreiranje kataloga i full-text indeksa nad kolonom Name (sa engleskim jezikom)



Primer 3.21. Full-text pretraga svih zaposlenih koji se zovu Milan korišćenjem metode CONTAINS

# **Zaključak**

U ovom seminarskom radu smo obradili internu strukturu i organizaciju indeksa. Na početku rada smo diskutovali o samom načinu skladištenja i arhitekturi DBMS-a, kao i o indeksnoj strukturi podataka kada su indeksi zasnovani na hešu i kada su zasnovani na strukturi stabla.

Pored teorijske osnove, detaljno smo se bavili svakim indeksom pojedinačno koje nudi Microsoft SQL Servera – sistem za upravljanje relacionim bazama podataka. Na praktičnim primerima smo videli kako indeksiranje utiče na brzinu obrade i plan izvršenja upita. Za svaki tip indeksa je prikazan način korišćenja, naredba kreiranja, kao i glavne njegove prednosti i slučajevi u kojima će najviše biti koristan.

Dizajniranje i implementacija odgovarajućih indeksa ključni su faktori za optimalne performanse SQL Servera, a izbor odgovarajućeg tipa indeksa zavisi od karakteristika podataka i zahtevanih performansi. Prilikom dizajniranja i upotrebe indeksa treba uzeti u obzir veličinu tabele, vrstu podataka, vrste upita koji se često izvršavaju, kao i količinu podataka. Uz pravilno dizajniranje i implementaciju indeksa, mogu se postići značajna poboljšanja u performansama baza podataka.

Pored toga, u novijim verzijama SQL Servera postoji podrška za memorijski optimizovane indekse, koji omogućuju još brži pristup podacima korišćenjem memorije umesto diskova. Međutim, potrebno je pažljivo razmotriti da li su memorijski optimizovani indeksi adekvatni za određeni tip podataka i zahtevane performanse.

Konačno, treba imati na umu da indeksiranje tabele može dovesti do većeg opterećenja sistema tokom ažuriranja, brisanja i dodavanja novih podataka, jer indeksi takođe treba da se ažuriraju. Samim tim je važno balansirati potrebu za brzim izvršavanjem upita i potrebu za održavanjem optimalne brzine rada baze podataka.

# **Literatura**

1. Gehrke J., Ramakrishnan R. (2003): *Database Management Sytems, Third Edition*, McGraw-Hili (SAD)
2. Pages and extents architecture, <https://learn.microsoft.com/en-us/sql/relational-databases/pages-and-extents-architecture-guide?view=sql-server-ver16>, (pristupljeno 1.4.2023.)
3. SQL Server Indexes, <https://learn.microsoft.com/en-us/sql/relational-databases/indexes/indexes?view=sql-server-ver16>, (pristupljeno 2.4.2023.)
4. SQL Server and Azure SQL index architecture, <https://learn.microsoft.com/en-us/sql/relational-databases/sql-server-index-design-guide?view=sql-server-ver16>, (pristupljeno 2.4.2023.)
5. Columnstore indexes, <https://learn.microsoft.com/en-us/sql/relational-databases/indexes/columnstore-indexes-overview?view=sql-server-ver16>, pristupljeno (8.4.2023.)
6. Indexes on Memory-Optimized Tables, <https://learn.microsoft.com/en-us/sql/relational-databases/in-memory-oltp/indexes-for-memory-optimized-tables?view=sql-server-ver16>, pristupljeno (9.4.2023.)
7. Spatial indexs, <https://learn.microsoft.com/en-us/sql/relational-databases/spatial/spatial-indexes-overview?view=sql-server-ver16>, pristupljeno (15.4.2023.)
8. XML indexes, <https://learn.microsoft.com/en-us/sql/relational-databases/xml/xml-indexes-sql-server?view=sql-server-ver16>, pristupljeno (15.4.2023.)
9. Full-text search, <https://learn.microsoft.com/en-us/sql/relational-databases/search/full-text-search?view=sql-server-ver16>, pristupljeno (17.4.2023.)